Generation of point cloud mesh and 3D design verification with LiDAR, depth camera and Cobots

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Valkeakoski
Degree Programme in Automation and Electrical Engineering
BEEAP17A7

Tuan Duong Truong

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Automation and Electrical Engineering
Valkeakoski

Author Duong Truong **Year** 2020

Subject Generation of point cloud mesh and 3D design verification with LiDAR, depth camera and Cobots

Supervisor(s) Katariina Penttilä, Khoa Dang

ABSTRACT

”The more we reduce ourselves to machines in the lower things, the more force we shall set free to use in the higher.”

Anna Callender Brackett

Applying automation in the production line is the groundwork of the development in modern industry and a primary trend in technical progress. Automation in production promotes the labor efficiency and the quality of manufactured products and enables an optimal use of all production resources. This thesis was commissioned by the Degree Programme in Automation Engineering and a concrete industry business to automate the precast concrete slab quality control process. The empirical targets for this thesis project included developing and commissioning of a system capable of operating automatically in reconstructing a concrete slab surface in 3D. In addition, the system was designed to be able to operate in various plant conditions and production processes by customizing the software and with an ability to integrate additional hardware.

In the project development phase, academic notion regarding to 3D restructuring solutions, related devices such as a camera, a sensor as well homogeneous programming environments were discussed along with concepts concerning data communications, the ability to operate independently and expand. In general, the concepts discussed focused on 3D reconstruction in a larger field, computer vision and computer graphics along with integrating IoT into use for remote operation capability.

The most important result of thesis project encompasses a realsense RGB-D camera designed for the ability to augment conventional images collected from the environment by depth information combined with SLAM algorithm to form a handheld 3D scanning system. Moreover, the system also had the ability to function flexibly when it was capable of operating as not mererly as a fixed scanning system but also a pre-

programmed scan orbit system with a robot arm. In addition, the scanning system also included an extended design to act as a horizontal planar scanner where it opens up a possibility to use 2D LiDAR to enhance the quality of scanning. Finally, All data communications between the devices in the system were implemented on the basis of the network infrastructure of a local WLAN intranet system allowing the system to operate remotely and independently from certain fixed hardware. The scanning object to verify the precise quality of the system's 3D model output were hollow slabs provided by the concrete industry business.

The objectives set for this thesis were met and the results accepted by the research unit of HAMK and the concrete company along with a consensus to continue developing the project with future plans.

Keywords 3D reconstruction, RGB-D camera, 2D LiDAR, Robot arm, IoT

Pages 84 pages including appendices

CONTENTS

List of Figures

1 INTRODUCTION

1.1 Background

The construction business, with an annual market capitalization exceeding 10 trillion dollars, is one of the most important economic sectors worldwide. (Bogue, 2018). The investment in construction industry accounts for between 9% to 15% of total GDP in majority of countries and national investment funds can disburse half of their investment in built environment. (T.D. Oesterreich, 2016). Despite the undeniable tremendous economic significance and astounding potential of the construction industry, it has not yet grown to its fullest, recorded in the fact that many companies are facing increasing due to the lack of skilled workers, sluggish and reckless productivity growth. Productivity in many other fields has heightened regularly during the past fifty years, which is not true of the construction industry as productivity in this area has rarely raised, even tend to decrease at some point. (JFCC, 2013)

The Eleventh ISARC symposium announcement says, "The use of automation and robotics in construction is a real prospect. It is happening now. Whilst the enabling technology is continuing to be developed and refined, this is no longer the most significant barrier to progress." (Chamberlain, 2 - 12, 2012). During the 4.0 technology era, robotics and automation systems are the key to the comprehensive revolution, bringing superiority for the construction industry and for the entire area of Architecture, Engineering and Construction (AEC) in particular and the whole human civilization in general (L.H. Forbes, Autom. ConStruct., 92 (2018), pp. 297-311). Construction is an industry that requires absolute accuracy. Manual construction methods have reached their limits and automation technologies have proven to be a pioneering solution in other areas to cut down labour costs at the same time improving productivity and quality, especially in industries that require high accuracy, are a new transition to solve defiance of productivity, accuracy and pursuit of automation of construction lines.

A number of participatory development projects have been started in Finland to create the necessary foundations for the fourth wave of industrialization in construction. The concrete industry business supporting this project leading concrete industry in Finland, with more than 65 years of development, has boldly pioneered the automation of production lines. With thirty one factories in total, the company provides the Finnish construction market with a wide range of precast and ready-mixed concrete products. The two products are hallow core slab and flue element, with the characteristics of precast concrete beams, which must have absolute exact specifications compared to the customer's design, has been paid much attention in the final quality checking stage. This

thesis is partially under “Point cloud mesh generation and 3D design verification with LiDAR, Depth Camera and Cobots” project, which is carried out by the cooperation of HAMK and a concrete industry company. As a part of the work, in this report, the following three key issues will be discussed sequentially, namely the method of scanning the surface of concrete beams with cameras and robotic arms, the ability to automate the scanning process including the operation and human tasks.

1.2 **Problem description**

1.3 **Aim**

The commencement of the project described in this thesis started from the hypothesis that an automatic scanning object surface system with the support of a sensor and a camera combined with the maneuverability of robot arm would be more feasible and effective than a manual measurement method. Two types of slab were used as the target scanning entity for this project are a hallow core slab and a flue element. The absolute uniformity between the actual product and the drawings is a peculiarity of precast concrete panels used in construction. Therefore, the 3D mesh generated in a scanning process with minor deviation was ideal to be used for the purpose of verifying whether the 3D design would satisfy the parameters of the original design of hallow slabs.

Realsense depth camera D451i and LiDAR TiM561 were considered as a suitable and optimal option in the project for slab surface scanning with minor deviation purpose. With the robot arm, a low cost UR5 robot arm was the solution for flexibility of the system. To support the hypothesis, sequential solutions were set up to provide specific solutions. These solutions were also the aims in this project. The first target was set out using low-cost RGB-D cameras for the design of a handheld 3D scanner. A combination of a 2D LiDAR sensor and a robot arm on the handheld scanner basis structure to improve the quality of the 3D model output of the reconstructing process and increase the stability as well as self-propelled of the scanning system, were considered as the next goals. In addition, the system was designed to commit the aim with criteria of decentralization and flexibility, using current powerful network infrastructure for data communications in the system so that any new devices or sensors could join in to transmit and receive data from the system, bypassing some barriers on hardware binding. Finally, it is equally important to conclude that all of these planned solutions aimed at the primary goal of developing an automated scanning system with a good quality output at a low initial investment, with potential to invest in both technical and business aspects.

This thesis is divided into six chapters as follows. The second chapter is used to review and discuss the theoretical concepts for the project solutions, which encompassed the basis notion of aspects of 3D

reconstruction solution in computer vision and computer graphic, commonly used devices in the 3D scanning field and their operating principle along with a homogeneous programming environment used in this thesis project. The next chapter focuses on introducing the equipment selected to be used in the project as well as its role in the larger whole as a versatile 3D scanning system. In addition, the programming environment in this project is introduced in this section with two aspects: data communications and ROS packages. Descriptions in detail of the solutions which were commissioned at the target, the structure of the software system solution and the author's reasoning for designing these solutions, are discussed thereafter in chapter four. Chapter five describes the results of the commissioning process and the preliminary 3D model. Chapter five discusses current limitations, as well as near-future goals for development continuation of this thesis in the following year, as well as near-future goals for development continuation of this thesis in the following year. Chapter four describes the results of the commissioning process and the measurement procedure made on real hollow slabs and on 3D models for preliminary assessments of the scanning system outputs. Chapter six discusses the results of this thesis, brief look at the potential of implementing the system in real plants, as well as goals for a continuation of this thesis in the near future.

2 METHODOLOGY

3D scanning is a fairly new concept when compared to modern technologies. However, the potential of this technology is unlimited. Over the past ten years, the cost of devices used in this technology has decreased, making them more accessible to larger consumer groups while quality is increasingly being improved. However, the price for a scanner is still very high. As mentioned in the introduction of this thesis, through this research project, an automated scanning system to reconstruct hollow slab surfaces in 3D environment with low initial investment costs was developed to be applied at the end of a production line to check the output quality of the products. In this part of the paper, we will introduce the technologies and methodology that were studied and used in this project. In this way, techniques, technologies, and terms are explained to help followers understand how our research project was concluded.

2.1 **3D reconstruction**

3D scanning technology is a procedure of reconstructing the surface shape of an object in three dimensions to create a 3D digital model. 3D scanning has opened a new turning point in 3D technology, any physical

model that exists in the world can be modeled with digital data in a short time. This technology is considered to have great potential in many fields from mechanical manufacturing, archeology, health to transportation and construction etc. Implemeting 3D scanning technology in assembly-line production will help businesses save time, reduce costs and improve output quality for products. The first usability of 3D scanning technology is reverse engineering. From a product object, 3D scanning technology will help recreate the model on the machine correctly. With digital models of products, businesses can improve designs to be ready for production. Almost every product has just been manufactured, it needs to be checked and reviewed to ensure it has met the requirements of the policy on the output quality of the business. In particular, especially for products with surfaces that require high accuracy, 3D scanning technology will help businesses reduce "anxiety" and "improve" quality in quality control, actualizing competitive advantage compared to market competitors. In general, the important role of 3D scanning technology in production is undeniable, 3D technology makes a great contribution to improving the quality of production processes and it promises that 3D scanner technology applications will increasingly be used more widely in the production segment of businesses in the near future.

When developing a self-propelled robot or a standalone 3D scanner with the goal that the system can operate freely in an un-mapped environment and discover new areas on its own, developers must consider three basic but most important factors that are localization of the system in the environment, mapping of surroundings and navigation. The relationship of these three factors is shown in the image 1.

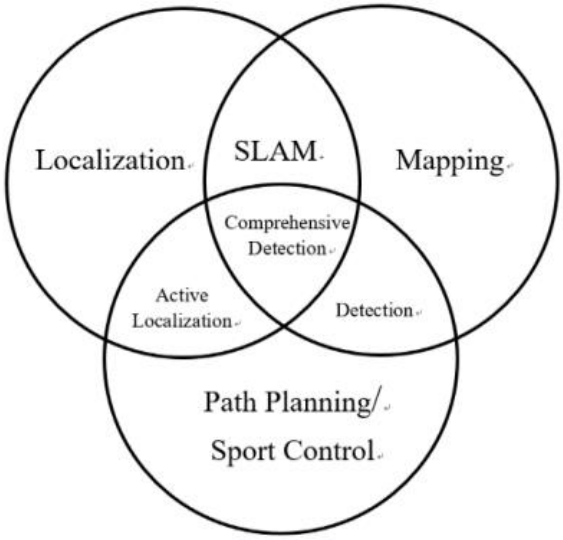


Figure 1. Relationship between the three most important factors in 3D scanning - localization, mapping and navigation

Localization and mapping are the key issues of Simultaneous Localization and Mapping. Simultaneous Localization and Mapping (SLAM) is

considered as a fundamental solution for robots to become truly autonomous. In the SLAM algorithm, a robot must perform two tasks simultaneously, which is to organize a map of the surrounding environment and to locate its own position in the environment map. This is what makes the SLAM algorithm so complicated, because it's not just localization or mapping alone, but both, and above all, SLAM's mapping and localization are not provided from the beginning makes the problem return to one of the perpetual questions: the chicken comes first or the egg comes before.

In other words, SLAM builds an environment map based on the location and pose of the robot and also determines the position and pose of the robot based on a known map. The method of operation is to create a map and whenever estimating the position of a robot is known with better accuracy, update the previously set map and repeat the whole process. This process can be executed by using sensors to perform an environmental map setting and determine position and direction. In an unknown environment that requires discovery, robots or scanning systems need to use sensors such as optical sensors to collect environmental information, use this information to build maps and perform updates to the old map according to the information received from the sensors. From the map created, combined with the environment information, the robot can determine its position in the map. The result returned from SLAM will be the localization value of the robot in the generated map.

Definitely, there are various approaches to allowing a system position itself in the environment such as installing a pre-installed path, or GPS modules and even marker-based plate preinstalled throughout the environment, etc.

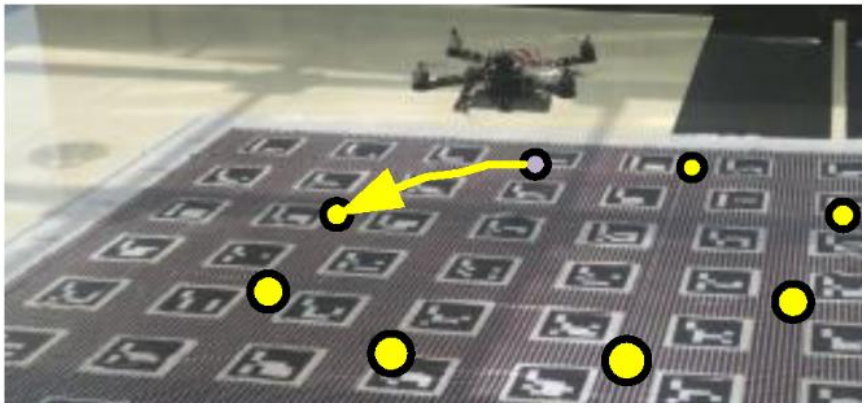


Figure 2. Marker-Based Multi-Sensor Fusion Indoor Localization System

However, these methods do not provide fully completed and independent autonomy for the robot system but also depend on many different factors such as the need for GPS connectivity in the area, the need to install the predefined path with a predetermined route. The alternative methods that can improve these concessions come from

simpler sensors but offer more reliability. Sensors that can be used to replace GPS or pre-installed tracks can be RGBD cameras or Laser sensors like LiDAR 2D, combined with Inertial Measurement Unit (IMU) such as IMU built into realsense D435i as the solution that was developed in this project. The common point of these sensors is that they do not collect location data directly, but system information in the environment is obtained through post-processing as generating odometry from laser scan or feature tracking from images flow. The environmental information gathered from these sensors indirectly provides the location of the system, different from the GPS system or track rails. Therefore, this solution can solve the problem encountered in unknown environments, thereby increasing the activeness of robot systems or 3D reconstruction systems. Because of the advantages of using highly portable sensors that indirectly collect data about the location of the system in the SLAM problem environment, our project will focus on developing SLAM methods by using the RGBD, LiDAR sensors and combine with the IMU to complete the 3D reconstruction of concrete slab.

2.2 **Visual SLAM (VSLAM)**

Basically, depending on the use of the sensor in different SLAM applications, we have different types of SLAM, but there are two primary types that are most popularly used: SLAM laser and visual SLAM, respectively with sensors being used are lasers and cameras. In this project, the RGBD camera is used as the core device for collecting environmental data for SLAM, although 2D Laser has also been used for positioning purposes but not for mapping purposes. Therefore, the methods of implementing SLAM proposed in this project are considered to revolve around VSLAM.

VSLAM is a technology for localizing a system and mapping in an undefined environment using the computer vision technology platform. Because VSLAM uses the technology of computer vision, only the cameras are required to use it, no other sensors are needed. However, it is important to determine that the camera used in VSLAM differs from conventional cameras, DSLR cameras. The cameras used in visual SLAM are capable of capturing images at high frame rates - the maximum frame rate for some Realsense cameras can be up to 90hz. These cameras can have different structures and functions, including monocular lenses, stereo cameras, etc. For conventional cameras, they simulate the surrounding multidimensional environment on a two-dimensional image strip, ignoring the depth of the environment. Similar to the human eye, because we have seen so many images, gradually our eyes get used to the actual distances of objects, allowing us to feel the depth in the environment. The human eye does not have the ability to perceive the depth of objects in sight, depth perception must be learned using an

unconscious reasoning. This can be demonstrated by the fact that there are still some cases in practice that deceive the human vision, which are based on estimates by natural intuition and habits. External factors unfamiliar to the human eye, such as the image below, can cause many people to have trouble determining which objects are near and which are farther away.

a) b)

Figure 3. a) Human vision may be deceived about the distance of objects in the image if the objects in the image have compositions that are unfamiliar to the human eye. b) and vice versa, an image drawn on a flat sheet of paper if it closely reproduces the image that the human eye is familiar with may deceive that the objects in the image have depth.

The cameras used in VSLAM will look at the environment in a perspective with both object and distance images, or depth. Thereby, we can determine the exact distance from object to object, from camera to object, as well as the actual size of the object. The structure as well as detailed operating principles of these cameras will be introduced in the next chapters of this paper.

2.3 **Imaging Environment**

In machine vision and graphic research in computer science, the term 3D reconstruction can be used to refer to a process of gathering images of the shape and size of an object in space. There are many solutions for 3D reconstruction but generally divided into two approaches: proactive and passive.

In order to reproduce a real object in 3D, many elements are required, the most important is the location of the scanning system in 3-dimensional space. Every system involved in reproducing the 3D environment uses sensors to determine their position. A 3-axis oxyz coordinate system is used to describe changes in the position of an object in space. The position and direction in the space of a solid body is defined as the position and direction of a main reference frame in another reference frame fixed to the object, translating and rotating with it (fixed reference frame of an object or its coordinate system). At least three independent values are needed to describe the direction of the fixed

frame. Three other values are needed to describe the position of an object. A freely moving solid is called 6 degrees of freedom (6DoF). In other words, in a coordinate system, the movement of an object is expressed by changing the position of a point in a certain direction, while the rotation of an object is expressed through rotation of a point around a fixed axis. (Daniilidis, 2016) (Hager, 2016)

Currently, there are many types of sensors with different modes of operation that can be used to collect variable input data streams, allowing to detect changes in rotation, displacement and even acceleration in motion thereby creating a 3D vision. Commonly used sensors can be listed as RGB-D cameras, LiDAR and sonars, etc. Next section of this article presents sensory devices used in computer vision and by most SLAM algorithms to capture environmental information, as well as basic information about estimation procedures.

2.3.1 Optical camera

In localizing in 3D environment and mapping, optical sensors are prominent device because of the technological benefits that they bring. Optical sensors designed to provide a high resolution image output with a high frame rate are very potential for many applications. In machine vision there are algorithms that can be used to leverage data from cameras to create 2D or 3D localization maps. In general, these cameras have a much lower initial investment cost than other sensors like LiDAR laser scanning sensors, sonars, etc (Bai, 2012). Everything has two sides, it has an advantage over other types of sensors in terms of market prices and power consumption and although the accuracy is rated high, it is still lower than the accuracy in the image data of more expensive sensors. Typically, RGBD cameras provide the 2-D rich texture image data while the higher-precision 3-D terrestrial point cloud data can be achieved with LiDAR, which has high precision but insufficient texture information. (X Kang, 2019)

2.3.1.1. Monocular Vision

Imagine you look at an object in the space in front of you, when you close one eye and still keep a view on the object, and then you close your open eye and open the other eye. Repeatedly changing your eye movement at a fast enough pace will give you the feeling that the object moves slightly when you change your eyes. And when you try to touch the object, you will find that it is a little difficult to locate the distance between your hand and the object. That is monocular vision. In general, in monocular vision, the visible area will be higher than binocular vision, but monocular

vision will be limited in depth perception between eye to object, or from object to object in space. (Torreao, July 19th 2011)

Monocular cameras are currently designed on the two most common types of camera sensor technology, Charge-Coupled Device (CCD) and Complementary Metal-Oxide Semiconductor (CMOS), of which CMOS is the most popular. On the surface of these two types of sensors are three types of diodes arranged alternately. These three types of diodes will capture three primary colors: Red, Green and Blue. Then these 3 colors will be mixed together to create the kind of colors that the human eye can see (Figure 4). For a CCD sensor, each time the diode operate, it captures light and converts it into electrical charges, which are converted into visual signals. CMOS sensor is slightly different, this type will use color filters type R, G, B then transfer data as digital signal through the wiring and then transfer to the memory card. Camera companies are now trying to make new sensor technologies based on the basic structure of CMOS.

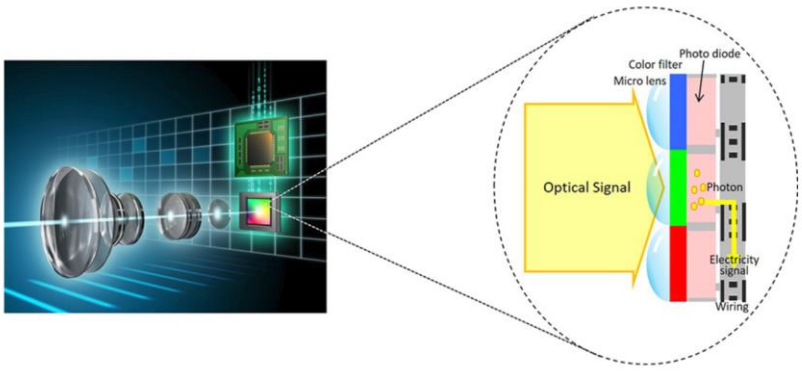


Figure 4. Cross-sectional diagram of image sensor (INK)

In the Simultaneous localization and mapping algorithm, the monocular camera acts as a camera to determine the angle. The mechanism of operation of the monocular camera is to let light reflected from the object through the lens and the image on the camera's film screen will receive the reverse image of the object. And because the use of this type of lens causes distortion of the resulting image, resulting in inaccurate image rectification, which is mainly used in machine vision-related applications. Image Rectification is to "edit" the images so that the epipolar lines are on the same row (same y-axis) for all images after projection.

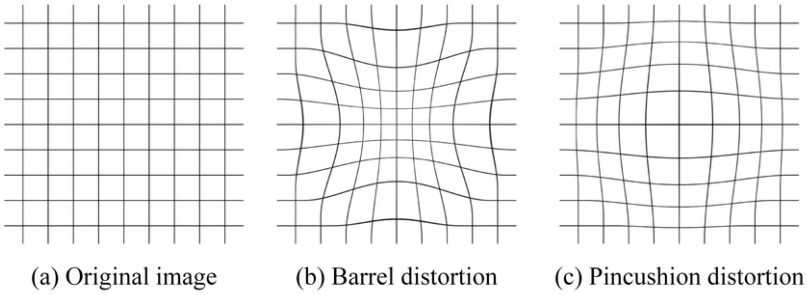


Figure 5. The types of distortions commonly encountered on projected images

FOV - field of view, showing how much of an image can be retrieved through the camera and lens used. FOV changes when the following two factors change: lens focal length and sensor size (or film size)

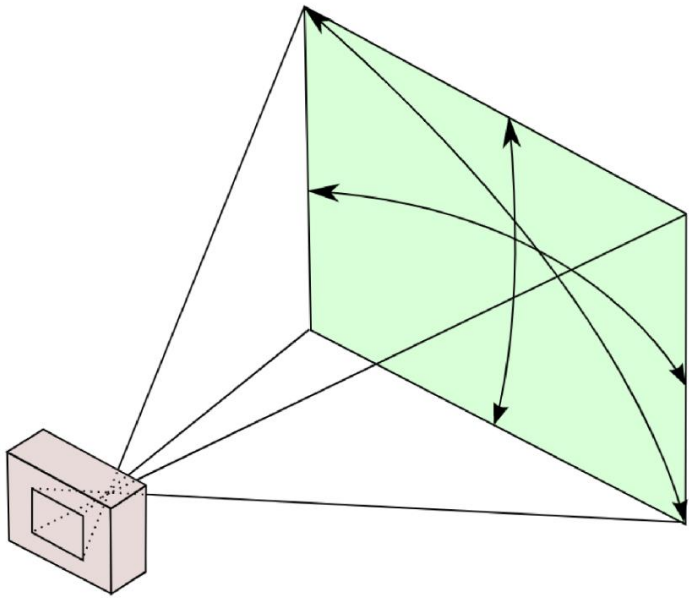


Figure 6. Camera Field of View

The larger the focal length of the lens, the narrower the area of view (open angle), otherwise the smaller the focal length, the wider the area of view. This is a physical principle rooted in the distance between the lens and the sensor (Figure 7). According to the operating mechanism of the monocular camera, the size of the field of vision is proportional to the distortion of the projected image.

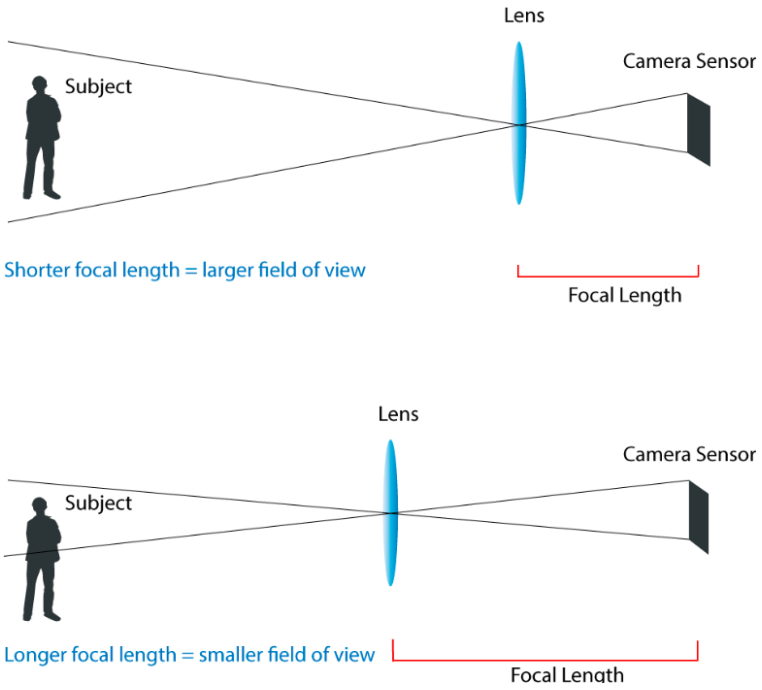


Figure 7. As focal length increases, the optical lens is moved further from the sensor making the field of view smaller

The Simultaneous localization and mapping algorithm using monocular vision is implemented based on feature recognition. With different lenses, the field of vision angle is also different. The larger the viewing angle, the higher the image area collected, the higher the number of features thereby helping the algorithm to operate more efficiently.

Figure 8. Photos from fisheye lenses (FOV <150) provide a greater number of features in the image than other types of lenses

2.3.1.2. Stereo camera

Stereo vision is a technique that uses two cameras to measure the distance between objects in space. The standard structure of stereo uses two cameras that lie on the same horizontal line at a distance from each other. The images collected from the two cameras will be analyzed to find common pixels. Mathematical rules will be used to analyze images such as congruent triangular rules combined with deviations of common

points to determine the distance or the so-called depth of object compared to the position of the camera. (Moqqaddem, Ruichek, Touahni, & Sbihi, 2011)

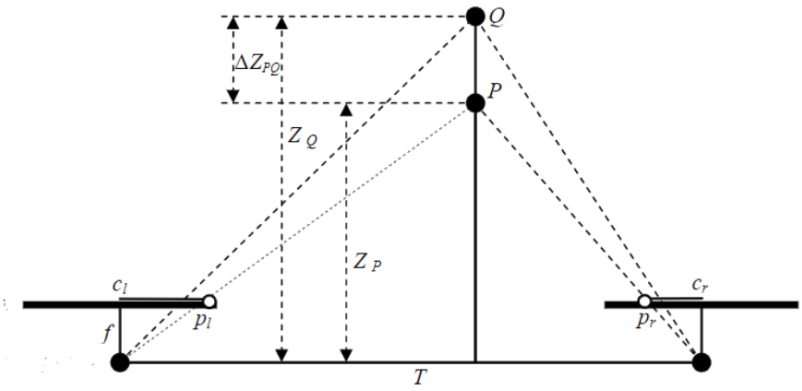


Figure 9. Standard structure of a dual-camera system with f focal length and T.

In three-dimensional space, the deviation d of a point P is determined to be the deviation between its projections on the two cameras shown in the figure above as pl and pr from their centers (Cl and Cr). Applying the properties in congruent triangles for the case in the Figure 9, we have the expression showing the relationship between deviation d and the distance Zp of point P with the camera:

$$d_p = p_l - p_r = \frac{fT}{Zp}$$

Thereby we have $Z_p = \frac{fT}{dp}$

However, finding the similarity in pixels of two projected images can sometimes be a problem, as the environment, especially indoors, often has insufficient image texture. In fact, the pixel is used to determine the deviation (i.e. not used in the metric system) and not necessarily from the center of the image (not always able to determine the center of the image correctly in reality).), so the relationship between the elements in the formula is shown in the following expression

$$Z_P = \frac{fT}{(d_P^{pxl} + d_0)w} = \frac{K}{d_P^{pxl} + d_0}$$

Where d_P^{pxl} is the deviation in pixels, d_0 is the constant associated with the eccentricity of the image and other errors if any, w is the width of the pixel and K is the constant $\frac{fT}{w}$, all in meter.

By applying the above formula, we can calculate the distance of the object from the camera from the deviation, from which it is possible to determine whether the two obscured objects are in contact with each other or simply a represent the object before another object by comparing their deviations or distances (for example, points P and Q in the Figure 9). Two objects come into contact when their deviations are approximately the same (or less than a threshold). In general, finding the contact state of objects is very important for the identification and reconstruction of objects in space. Depth data in 3D reconstruction are represented by a range of colors ranging from white to gray. White shows high intensity meaning that the closer the object is, the more color it tends to turn gray and black. From the distance (depth) data in space, combined with the RGb image from the camera, we have the color image output with the depth shown in 3D.

2.3.1.3. RGB-D camera

Camera RGB-D is the type of camera that uses two types of sensors simultaneously: conventional color image sensor like traditional camera types, for output image is RGB image and a depth sensor, for output image is depth image. The purpose of the additional depth sensing technology to the stereo camera is to detect distance where there are no features. The RGB-D camera range sensor system can capture or synthesize depth information on each pixel and combine it with RGB pixels, to create a complete depth map. In other words, RGB color images and Depth depth images, through processing steps creating 3D data in the form of point clouds. The point cloud is a set of points in three dimensions, each of which includes its XYZ coordinates. In addition, each point can also contain additional color information. In general, point clouds are the type of data obtained from 3D scanning devices. These devices collect information about the surface of objects on the principle that they emit a beam of electromagnetic waves (infrared or laser) and receive reflected waves. The result of the measurement from the scanner is a data set of points collected, in the form of point clouds.

The RGB-D sensor is also a form of 3D scanner when using the depth sensor according to the scanning principle and combined with the color sensor. In addition, point cloud data can also be generated from 3D models such as CAD models. Point cloud type data is used in robots and multi robots with RGB-D sensors, or the remote sensing industry with 3D terrain scanning devices using scanners mounted on drones. There are many different types of depth sensing techniques used in RGB-D cameras: light with a stereo structure or a stereo structure with flight time. (Gerald Rauscher, 2014) (M. Labbé, 2018)

Basically, RGB-D scanners have many advantages over laser scanners like LiDAR in terms of price, power consumption as well as weight and size. Currently, there are many types of RGB-D cameras on the market coming from different manufacturers with competitive prices, including Intel's realsense camera, Kinect camera of Windows, Xtion camera of Asus, etc. However, RGB-D cameras also have inherent disadvantages that they actually produce very accurate image data at close distances, about 5m, but the quality will gradually decrease as the distance from the object to the camera increases. In addition, the Field of vision of current RGBD camera types available on the market is quite small, ranging from 60 to 70 degrees. (P. Henry, 2010)

2.3.2 Range sensing camera

Ranging camera is a distance measuring sensor. Two techniques for estimating distances of ranging cameras are commonly used: triangulation method and flight time (ToF) (Robert B. Fisher, 2016). ToF sensors are often used for long-range applications because they require very precise timing sources and often provide modest resolutions. The flight time scanner measures the time between the emission of the laser beam and the reception of the return jars, in combination with the speed of the laser to calculate the distance from the camera to objects in the air space. Another type is the phase shift scanner, by comparing the phase difference between the emitting beam and the reflected beam to obtain the phase difference between them and thereby calculating the delay time. These two types of scanners are used for applications that have a distance from the camera to objects at medium and long range. In addition, triangle sensors can make depth measurements at a high level of accuracy and dense at relatively close distances, so they are widely used in the field of 3D reconstruction. They are based on the principle of a one-point triangle on an object from source and physically separate optical detector. Each method has a number of advantages and disadvantages, which will be covered in the next two subsections.

2.3.2.1. Triangulation method

In general, the triangle measurement method is based on knowing the angle of the rays of light reflected from a point on an object in space to the lens of two optical sensors. These two optical sensors are placed parallel on a straight line and are spaced at a certain distance. Another variation of a sensor that uses a triangular scanning method includes a light projector and an optical sensor or a camera, which works according to the principle that the scanned object distorts the light lines from light projector, this light distortion or curvation is captured by the camera and used to calculate object depth, structure and details. Two optical sensors spaced on a baseline encounter some disadvantages with scanned environments containing insufficient features. The method of using a

light projector will be governed by the intensity of the external light, for instance, from the sun or electric light, which can affect the beam emitted by the light projector as well as the light reflected off the optical lens.

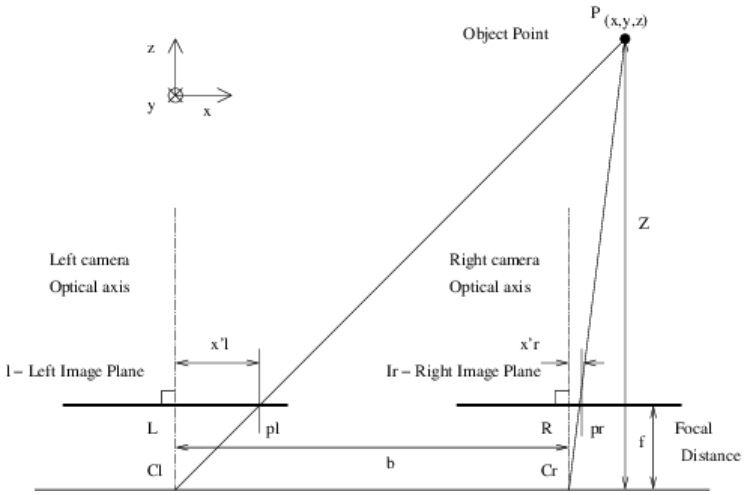


Figure 10. Optimal geometry for stereo vision

The image above describes Optimal geometry for stereo vision, where the distance between the two cameras is denoted as b. The deviation θ between the light beam from the second camera is denoted by Z and the baseline from the first and the second cameras can be calculated by applying a geometric formula $\tan\theta = \frac{z}{b}$ (Robert B. Fisher, 2016)

If the offset difference is denoted by x, the angle between the image plane and the main ray from the sensor is denoted and if the image plane is perpendicular to the camera baseline, the angle can be calculated following fomular $\tan\theta = \frac{f}{x}$ where f is Field of vision. From there we can find the height of the triangle in geometry or the depth of the object in space by expression $Z = \frac{fb}{x}$

On the other hand, one of the leading factors that determines the quality of a distance-measuring camera is the depth measurement error. We have the following expression $\frac{dZ}{dx} = \frac{-Z^2}{fb}$

Through this expression, we can see that the accuracy of the depth measurement of the camera is directly affected by the distance from the camera to the object in space and reducing the distance between the two cameras or Field of vision will reduce error in measuring depth. In fact, the triangle measurement method works with small errors when the distance from the camera to the object is a few meters, although, at longer distances, the error in depth measurement can be reduced by image processing algorithms and many different methods. In addition, changing the field of vision of the camera and the distance between the two cameras or between the light projector and the camera will be able to help reduce errors even though the scanning range as well as the scanning distance may be reduced. Increasing the baseline (b), for example, will increase the difference and thus increase the accuracy for

further objects. However, increasing the baseline will also cause a larger frame in each camera where the pixels cannot be matched. The common image plane moves away from the camera and the depth of field for close-up vision becomes poor. Increasing FOV (c) will result in a larger general plane but will reduce depth accuracy for further objects, because there is a limited number of pixels distributed over a wider FOV. Increasing FOV will also distort the image with wide-angle lenses, requiring more accurate calibration to obtain acceptable triangular measurement results.

2.3.2.2. Time-of-flights method

A lot of people confuse Structure Light and Time-of-flight as one, sometimes a manufacturer integrates into the camera system both Structure Light and ToF sensors. The ToF approach is understood as a calculation to find the distance between an object's surface and the sensor (or camera cluster), based on the time that light travels from the source to the object's surface and then returns. The key module for the mandatory ToF solution is the lighting source, usually LED or laser. Besides, of course the image sensor of the camera. Companies developing this technology include Microsoft, Sony, Infineon, and PMD Technologies. Because the sensing process as well as the calculation method of ToF technology is very fast, the distance calculation is highly accurate, this is a popular choice for applications that need to determine the direction of movement, collecting data by distance measurement.
The method of infusion time can be illustrated in the figure 11. A duplex or laser beam is emitted (emits only a few cycles) and reflects back from the object in space to the receiver located close to the transmitter. The transmitter and receiver can be integrated on the same sensor. The receiver can also be attached to objects. TOF is the time between the start of emitting and the return of the signal. The distance is determined by the formula $d = c.\frac{ToF}{2}$ when the transmitter and receiver are in the same position, and $d = c.ToF$ when the receiver is attached to the object. The precision is usually equal to ¼ wavelength when a return signal is detected, at the time its amplitude reaches the threshold. The amplification factor is automatically increased in proportion to the distance to ensure accuracy. Accuracy can be improved by the method of detection of the maximum amplitude. This makes determining the arrival time of the wave less dependent on signal amplitude. Ultrasound, duplex radio, or optical power sources are commonly used. So the parameters involved in calculating the distance are the speed of sound in the air (approximately 0.305 m / ms), and the speed of light (0.305m / ms).

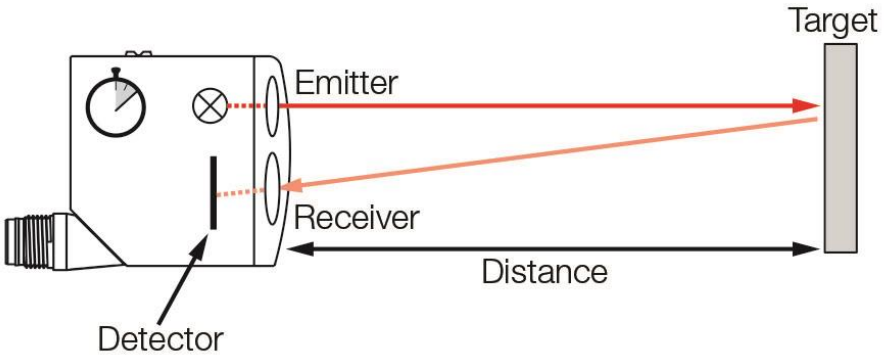


Figure 11. The laser is projected to the object from the emitter and the reflected laser is received by the receiver

Uncertainty, also known as time-walk error caused by a change in the return signal strength (is the result of a change in the reflectance of the object's surface, and the signal attenuation in the fourth power of distance due to spherical divergence). The difference in the returned signal density affects the rise time (which is the time for the signal to increase from 10% to 90% of the maximum amplitude value) of the detected pulse, and in the case of a fixed threshold detector, objects with low reflectance will appear at longer distances. For this reason, time constant separators are often used to set the detection threshold at a specified number of values, which are calculated according to the peak value of the received pulse. (Vuylsteke & Oosterlinck, 1990)

In terms of baud rate, we have a relatively low rate of sound propagation in the air. This is a feature that makes the ToF method the most commonly used method that prevails in low-cost audio systems. In contrast, the transmission rate of electromagnetic energy may impose some requirements for integrated control and measurement circuits in optical or radio frequency applications. Therefore, ToF sensors rely on the speed of light to require a timing circuit at nanoseconds to measure distances with a resolution of about one foot. (Drunk, 1988)

Laser distance measuring system ToF, also known as the laser radar or LiDAR, was invented in the early 1970s Laser energy is short, high-speed pulses that are emitted toward space objects, which need measuring distances. The total time it takes to reach the object and the time it takes to measure the distance, the speed of the gun is calculated by the speed of light transmission in the air. And because sensors use the ToF method to measure the length of time light travels, their distance measurement results will have a constant accuracy. This can be considered as an advantage of sensors using the ToF method using triangulation sensors, which have errors that are proportional to the measurement distance, usually increasing follow the quadratic equation. Simple ToF sensors include the LiDAR 2D, which can be used as a matching laser scanning device to determine the relative position of a system in two-dimensional space.

2.3.2.3. LiDAR

From the point of view of the components of a manipulator, LiDAR system is considered as a sensor. The term LiDAR stands for phrase “Light Detection And Ranging” used to a remote exploration technology that uses electronic radiation. Therefore, basically LiDAR system is a basic laser system that operates on the same principle as radar system or ultrasound waves.

We can explain the principle of operation of LiDAR a figure 12: The laser source - transmitter (1), emits a high beam towards the target and records the reflected light back by an optical radiation probe placed behind the monochrome filter. The laser beam direction emitted from transmitter can be changed by combination of a rotating mirror (2) inside the LiDAR device. The sequential order of laser pulses is synchronized with the rotation frequency of the motor of the rotating mirror and the desired angular resolution. The optical signal redirected by the rotating mirror is collected by detector after went through focusing medium (4). These signals are converted into an electrical signal. After that, the electrical signal is sent to the pulse filter, via an analog signal converter to a digital signal (ADC), digital signals are processed by a computer (MVT).

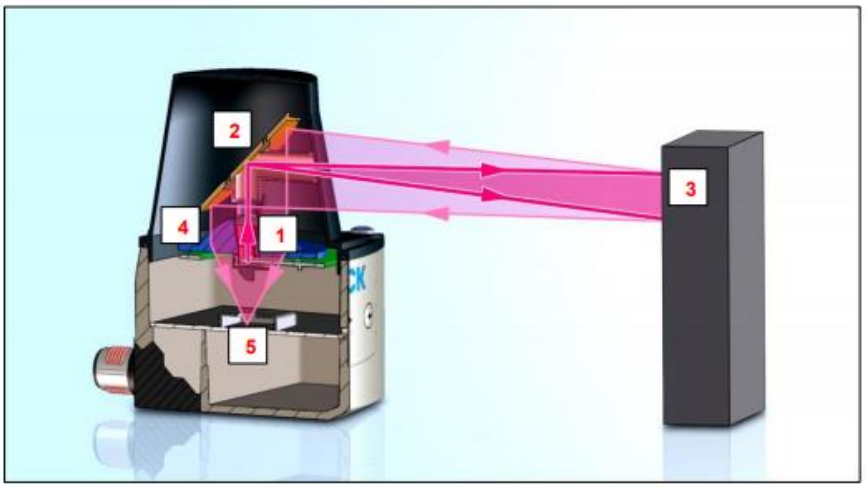


Figure 12. The structure of a LiDAR sensor

Discussing briefly about the definition and the basic working principle of the sensor, we know that sensors are used to capture physical changes in the surroundings over time interval, then convert them into uniform electrical signals. The following figure shows an overview of the way the current sensor works. In general, sensors measure and convert non-electrical variables (X_e) into electrical variables (X_a). These signals will usually be preprocessed before being forwarded to execution systems, these systems will have their own design and working principle depending on type and application of sensor.

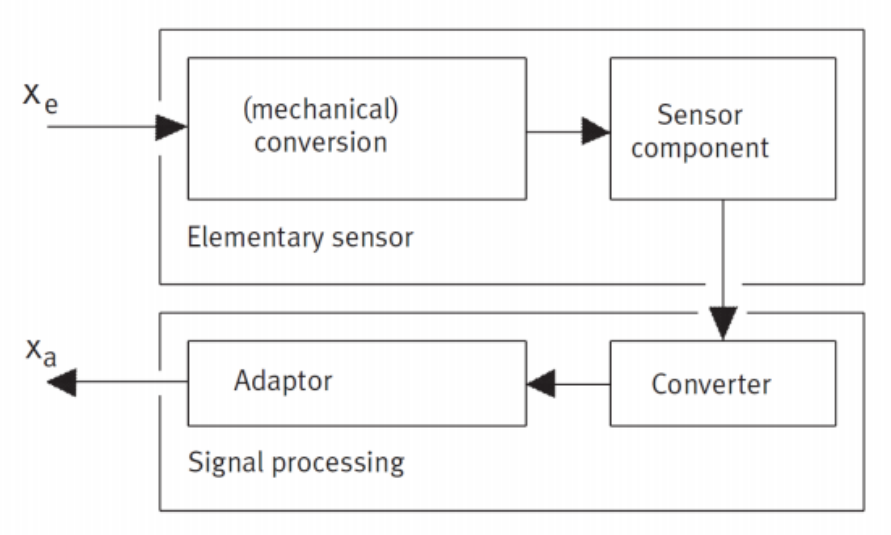


Figure 13. Sensor principle

The Lidar system determines the coordinates of the points in three dimensions X, Y, and Z by measuring the length D of the laser, determining the azimuth angle of the scan (based on the device's rotation angle and angle The rotation of the scanning mirror is determined by the IMU system) and the coordinate system selected at the time of laser scanning. Lidar equipment can have a range of tens of meters to hundreds of meters depending on many factors, especially thanks to the rotation angle of the mirror attached to the top of the laser emitting device towards the terrain surface. Lasers operate on the principle of electrical impulses with frequencies as high as a few Khz. Once created, the energy will be reflected from the terrain and objects through the optical system to the electrical impulse receiver. Based on the T time difference between the transmitted signal and the received signal, we can determine the laser length D at the time of scanning according to the formula:

$$D_i = T_i \frac{C}{2}$$

In which:

- D_i is laser length
- T_i is the time from the time the laser is emitted to the time the signal is received
- C is the speed of light

Lidar devices operating in the near-infrared spectrum with a wavelength of about 1504nm allow determination of length D with high accuracy with an error of about ± 1cm

(Silfvast, 2004)

2.4 **Robotic Systems**

According to the ISO organization's definition: “an industrial robot is defined to be an automatically controlled, reprogrammable, multipurpose manipulator, programmable in three or more axes, which

can be either fixed in place or mobile for use in industrial automation applications." (International robot standardization within ISO)

The general structure of an industrial robot includes manipulator, power supply, actuators and controller. Regarding to manipulator, manipulator is a mechanical structure consisting of links and joints. They are combined forming the arm to create basic movement.

- The wrist helps the arm trajectory more ingenious and flexible, and the hand or the end effector directly complete the operations on the object.
- Sensor system includes sensors and other signal conversion equipment. The internal sensor system of the robot arm allows the robot to recognize the status of the devices and the external sensors keep track on the status of the environment.
- The actuator: executing the movement for the link of robot arm. The power source of actuators is the engine of all kinds: electricity, hydraulic, pneumatic or combined.
- Controller: is usually a numerical control system with a computer that monitors and controls the operation of the robot.

Main components of an industrial robot are shown in Figure 14 below:

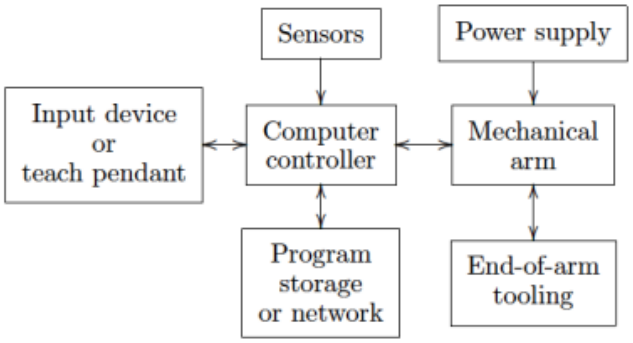


Figure 14. Components of a robot. (Mark W. Spong, 2005)

Kinematic manipulator:

Kinematics is the study of executable movement of a robot. The concepts researched in science of kinematics are the velocity (V), the acceleration (A), the position and higher order derivatives of the variables of position. Mentioning about kinematic in robotic, we are discussing about robot motion of the geometrical and time-based concept. In facts that, by using different types of joints such as prismatic and revolute joints, various kinematic chain could be formed. However, only few types of manipulators are used frequently in practice, which will be represented in this section.

Classification by structure:

- Two forms of primitive motion are used as the standard: - straight motion in the directions of X, Y, Z in normal three-dimensional space creates angular shapes, called Prismatic (P).
- Rotation movement around the axes X, Y, Z denoted (R)

Articulated manipulator (RRR): An industrial robot designed with rotary joints is known as revolute or anthropomorphic manipulator or articulated manipulator. An anthropomorphic robot could be formed from the composition of two to more than 9 joints. With the manipulators have three rotary joints, the axis of the first rotary joint is installed in vertical, the second and the third joint are placed in parallel. The most preeminent of articulated robots is relatively large movement freedom in a compact space. (I.J. Nagrath, 2003)

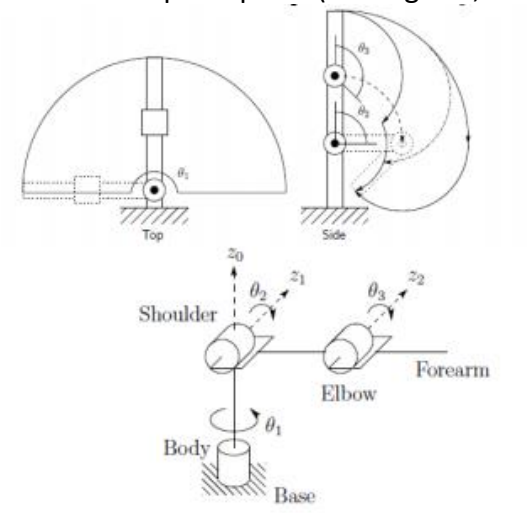


Figure 15. Articulated manipulator (RRR)

Spherical manipulator (RRP): Similar to articulated manipulator, however, the only difference is that the third revolute joint is prismatic joint. In the other words, a spherical manipulator is a robot with two of revolute joints and one prismatic joint. A spherical robot is also known as a polar manipulator, which is formed from three mutually perpendicular axes. It is important to note when using this type of manipulator is that the joint variables are the spherical coordinates of the end-effector with respect to the base. (I.J. Nagrath, 2003)

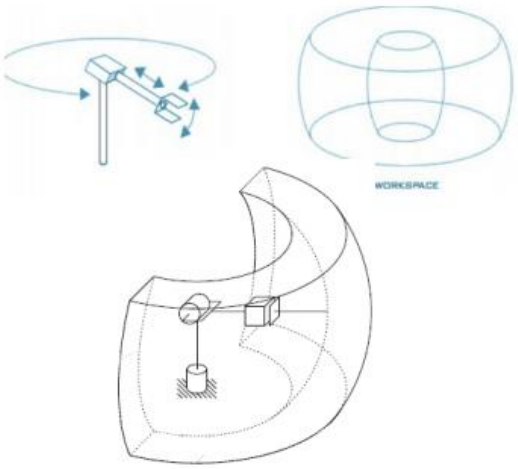


Figure 16. Spherical manipulator (RRP)

Scara Manipulator (RRP): SCARA is stand for Selective Compliant articulated robot for assembly. A scara robot is a spherical manipulator with parallel axes, meaning that scara robot is suitable for assembly operations. A scara robot design allow it to perform tasks that require meticulous and precise tracking and quick assembly or packing. (I.J. Nagrath, 2003)

Cylindrical manipulator (RPP): A cylindrical robot has the first rotary joint performing a rotation movement around the robot base, the second and the third joints are prismatic. The joint variable in cylindrical manipulator is defined as the cylindrical coordinates of the end effector with respect to the base. The construction of cylindrical manipulator create a solid structure, leading to ability to carry heavy payloads in a good scale of repeatability. (I.J. Nagrath, 2003)

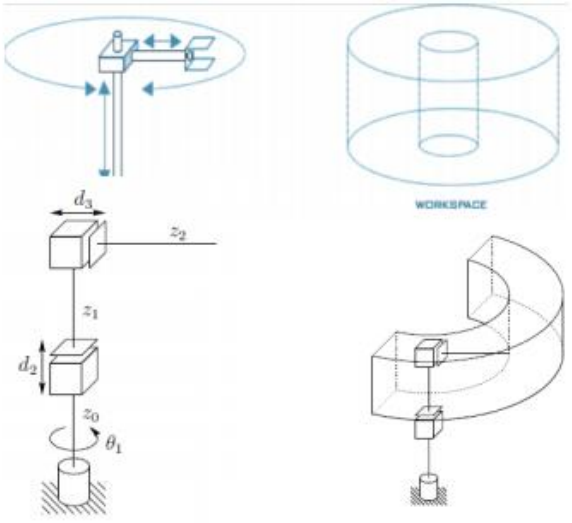


Figure 17. Cylindrical manipulator (RPP)

Cartesian Manipulator (PPP): In the industrial robot, rectangular or rectilinear or gantry robot is also known as Cartesian robot. Rectangular manipulator is design with the first three prismatic joints allowing it to reach the end effector gripper to most of position inside the cube or cuboid workspace. The joint variables in Cartesian robot are the Cartesian coordinates of the end effector to the base. Thanks to the structural characteristics, a rectilinear manipulator is suitable for transferring of materials and on-table assembly. (I.J. Nagrath, 2003)

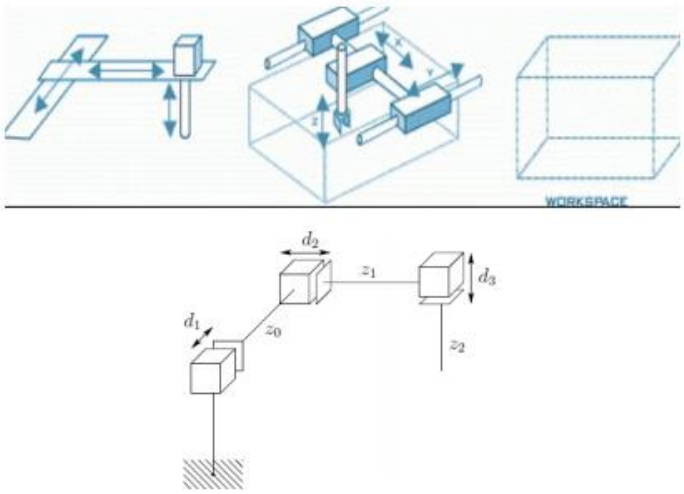


Figure 18. Cartesian Manipulator (PPP)

UR5 robot construction is designed with five rotary joints, meaning that UR5 robot is considered as an articulated manipulator. Details of the structural characteristics of UR5 robot are presented in the following chapter.

2.5 Modeling of robot

Rigid motions are defined to be an ordered pair (*d,R*) in which d stands for translation vector $d \in R^3$ where *R* represents for rotation matrix of the Special Orthogonal group of order three $R \in SO(3)$. For any $R \in SO(n)$ we have the general properties by convention of rotation matrix that *det R = 1* and rotation matrix has the properties of an orthogonal matrix, reflected in $R^T = R{-}1$ equation. Rotation matrix can be used to define the orientation of one coordinate frame according to the another, and the coordinates transforming from one to the another. Continuous rotations as rotational transformation of the $O_iX_iY_iZ_i$ frame to $O_jX_jY_jZ_j$ frame and to $O_kX_kY_kZ_k$ frame can be obtained by the equation:

$$Rki = RjiRk$$

Homogeneous transformation makes working with complex rigid motions simpler by cutting down the composition of rigid motions. A typical

homogeneous transformation matrix with $\boldsymbol{A} \in \mathrm{R}\ 4\mathrm{x}4$ is defined with a equation format of: $A = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}, R \in SO(3), d \in R^3$

Associating with the properties of that rotation matrix has the properties of an orthogonal matrix, we have a corresponding equation is

$$A^{-1} = \begin{bmatrix} R^T & -R^T d \\ 0 & 1 \end{bmatrix}$$

(Mark W. Spong, 2005)

2.5.1 Homogeneous transformations

The convention of point matrix written after transformation matrix has the transformation matrices as follows:

$$\mathrm{Rot}(z,\alpha) = \begin{bmatrix} c_\alpha & -s_\alpha & 0 & 0 \\ s_\alpha & c_\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$\mathrm{Rot}(y,\beta) = \begin{bmatrix} c_\beta & 0 & -s_\beta & 0 \\ 0 & 1 & 0 & 0 \\ s_\beta & 0 & c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$\mathrm{Rot}(x,\gamma) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_\gamma & -s_\gamma & 0 \\ 0 & s_\gamma & c_\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Trans}(M,N,P) = \begin{bmatrix} 1 & 0 & 0 & M \\ 0 & 1 & 0 & N \\ 0 & 0 & 1 & P \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because these four matrices can represent the motion of any object in space, the orthogonality of the rotation matrix performing in a homogeneous form is not guaranteed. (Mark W. Spong, 2005)

2.5.2 Kinematic

2.5.2.1. Position and orientation of solid objects in space

Coordinate system of object:

A solid object in space can be completely determined if its position and direction are described in a given frame of reference. Figure 19 is the standard O_{xyz} coordinate system with unit vectors x, y, z, used as an original reference frame. Normally, to represent and determine the direction of a solid object in space, we have to put it in a local frame of reference, for instance a reference frame *O'x'y'z'*. The original coordinate of this frame represents for the position of the object in the root Oxyz reference frame, the follow equation shows their relation:

$$O' = o'_x x + o'_y y + o'_z z$$

In which, o'_x ,, o'_y ,, o'_z are the perpendicular reference of the vector O' on the Oxyz coordinate system. The position of point O' through vector O' can be descripted as:

$$o' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}$$

The direction of the object is represented by the unit vector x', y', z' of the reference frame O'x'y'z', the relationship between them is described in the following expressions:

$$x' = x'_x x + x'_y y + x'_z z$$
$$y' = y'_x x + y'_y y + y'_z z$$
$$z' = z'_x x + z'_y y + z'_z z$$

The components of unit vectors (x'x, x'y, x'z) is cosin direction of the axes of the local coordinate system compared to the general reference system.

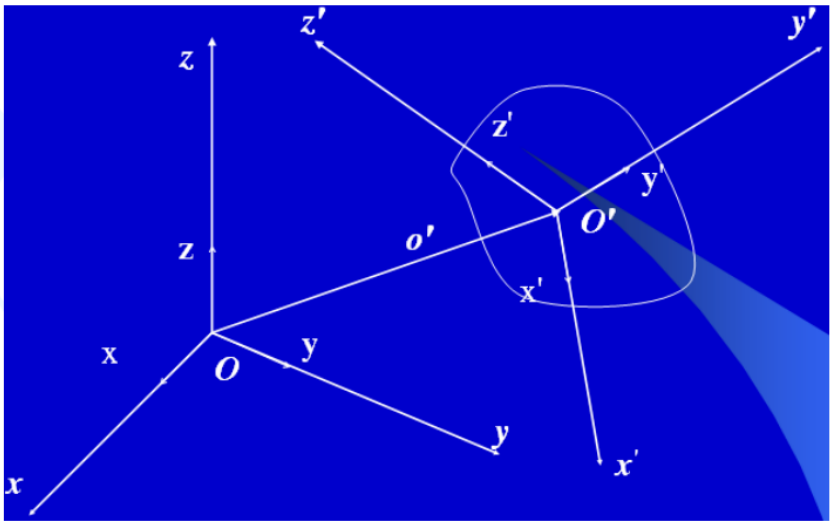


Figure 19. The position and orientation of a solid object are shown in space

Rotating matrix:

Basically, we can shorten the three vectors representing the direction of the object as above by representing them in a (3.3) matrix form, called rotating matrix:

$$R = [x' \quad y' \quad z'] = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} x'^T x & y'^T x & z'^T x \\ x'^T y & y'^T y & z'^T y \\ x'^T z & y'^T z & z'^T z \end{bmatrix}$$

Rotation around a coordinate axis is a special case of rotation of an object around a three-axis in space, the direction of rotation is conventionally set to be positive if the direction from the top to the origin coordinates of the reference frame under consideration is counterclockwise.

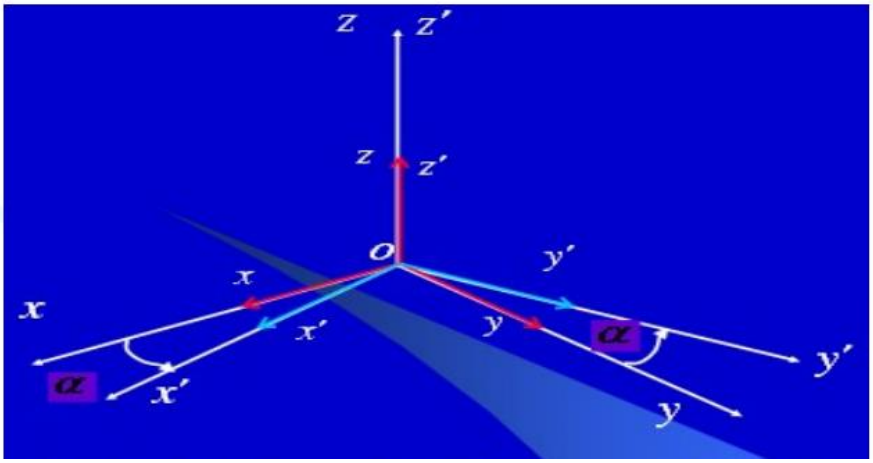


Figure 20. Rotating O-xyz aroung the axis z

Assume that the frame of O'x'y'z' reference formed by rotating reference frame Oxyz around an axis z at α angle, unit vectors of this system can then be represented in the frame of Oxyz reference as follow:

$$x' = \begin{bmatrix} \cos\alpha \\ \sin\alpha \\ 0 \end{bmatrix}; y' = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \\ 0 \end{bmatrix}; z' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

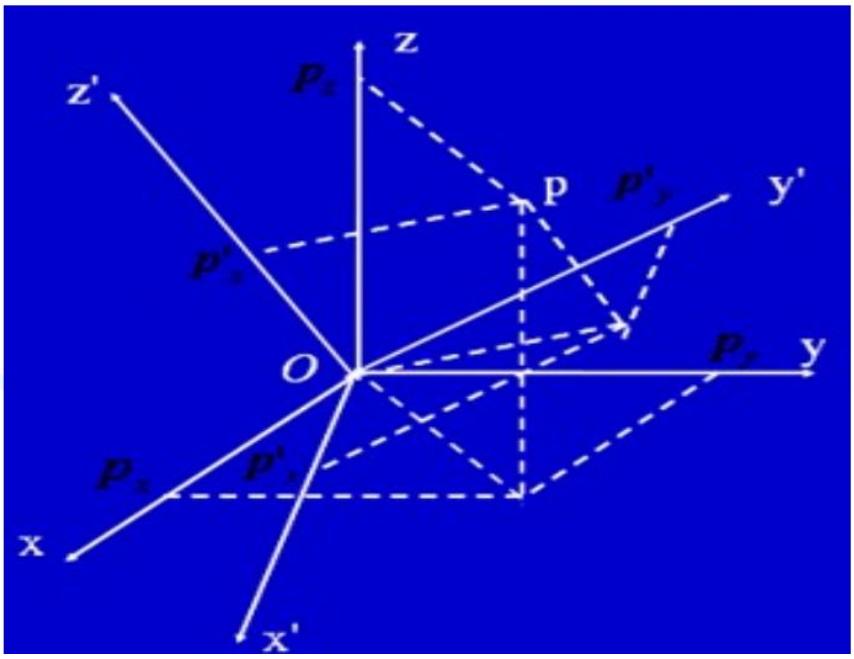
Respectively, the rotations around the axis z, y, z of O' reference frame compared to O reference frame, formed as:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} R_x(\gamma)$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

Rotations from the basic rotations around the axes of a frame of reference allow set up matrices rotate an object around any axis. These matrices are orthogonal, we can determine its inverse in two ways are changing the sign angle to the rotation matrix or transposing the rotating matrix. (Mark W. Spong, 2005)

Rotate a vector:

We can describe the rotation of a vector using the rotation matrices mentioned above, the following is a description for point P in two reference frame which have the same origin.



We take turns to describe the points in the two coordinate systems and then proceed to unify the two coordinates as follows:

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}; p' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix}$$

And because we are describing the same point, we have the same identity as follows:

$$p = p' = p'_x x' + p'_y y' + p'_z z' = [x' \quad y' \quad z']p' = Rp'$$

This equation can be transformed into new equation:

$$p' = R^T p$$

If written in rotation matrix with the full form of rotation, we have the following matrix:

$$p = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} p'$$

where the columns of the rotation matrix are cosin direction of the corresponding pairs of axes between the two reference systems. And because the structure of a frame of reference is perpendicular to each pair so the nine components of the rotation matrix are only three really linearly independent (Mark W. Spong, 2005)

2.5.2.2. Rotating a vector around a axis

Summary of rotation matrices:
In the process of identifying objects in space, rotations may not be performed on a base axis which is the basic axis of the reference system, but random axis. Therefore, to perform a rotation, we need to know two basic points:

- The rotation around any given axis may be equivalent to the many rotations around the basic axes of the frame of reference, where each rotation around the basic axes of the frame of reference is characterized by the corresponding Ai matrix of the given form. as above.
- The representation of a series of rotary transform operations is performed by successively multiplying in the order of the matrixes typical for each step.

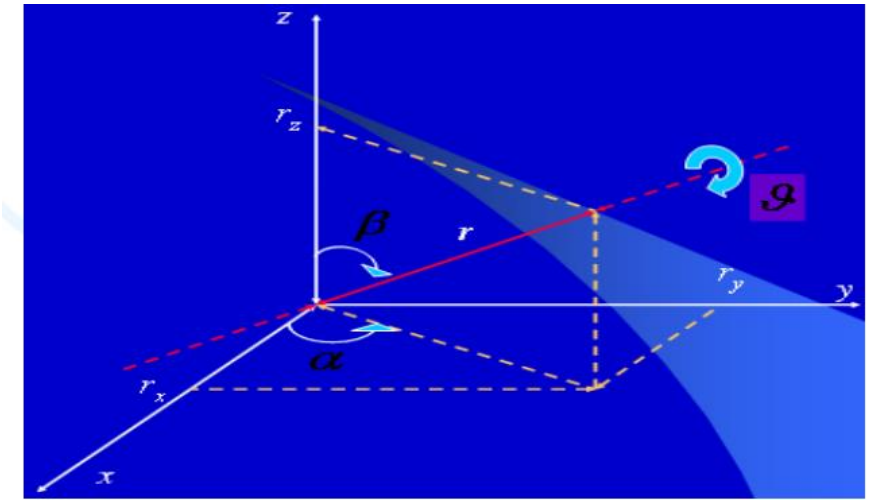
If the symbol P^i is the point P represented in the frame of reference i, it is also an expression of the rotation matrix of system i compared to system j. Consider the following relationship chain:

$$P^1 = R_2^1 P^2$$
$$P^0 = R_1^0 P^1$$
$$P^0 = R_2^0 P^2$$
$$R_2^0 = R_1^0 R_2^1$$

(Mark W. Spong, 2005)

Rotation on an axis:
This is a common case when describing mechanical dynamics, the idea of how it works is as follows:
1 -Change the axis of rotation compared to the reference system (or change of reference frame with respect to the axis of rotation) by the standard rotation matrix presented above so that the line acting as the axis of rotation coincides with one of the three basic axes of the system in the reference, call A1 the matrix used in this step.

It should be noted here that if the transformation of the rotating axis remains the same, the matrix A_1 is the standard matrix presented above, and if the transformation of the reference axis is compared to the fixed axis, the matrix $A1^T$ must be used, which is the displacement (inverse) of the standard rotation matrix. The A_1 matrix in the general case is always the product of two standard rotation matrices around 2 of the 3 basic axes of the reference system. The line v in the figure plays role as the axis of rotation, does not represent the rotating object that will be used as a basis. Because v does not coincide with any of the basic axes of the Oxyz frame of reference, it is considered to be any axis. However to describe v must know in advance as shown. To get v to coincide with one of the three basic axes, we can do the following:



We have A_2 = Rot(z, - α) is the matrix rotates v around the z axis in a clockwise direction when viewed from the top to the origin of the z axis. The purpose of this step is to make v coincide with the XOZ plane.

In the XOZ plane, we have A_3 = Rot(y, - β) is the matrix rotates v around the y axis in a clockwise direction when viewed from the top to the origin of the y axis. Now we have that v has coincided with Oz, the rotation around v has coincided with Oz being the aforementioned rotation. So, the operation of converting v to coincide with Oz actually consists of two steps as follows: A_1 = A_2 A_3. It can be concluded that to return v to coincide with the Ox or Oy axis also includes only two similar operations, and the angular data described v above are enough whether bringing v to coincide with any axis.

2 - Once any axis has coincided with one of the three basic axes of the above reference frame, the A4 matrix, which is the standard rotation matrix, can be used to perform the rotation around the v axis (this is now the basic axis).

3 - Return the results to the old reference system by doing the opposite of what was done in step 1, the inverse transformation matrix is the displacement (or inverse) of the positive change matrix.

For instance, to return the results to the old frame of reference: turn counter-clockwise around Oy axis by A_3T matrix following by turning counter-clockwise around Oy axis by A_3T matrix. So the whole process of describing an angular rotation of an object around any v axis, is a general matrix of changing steps that the sequence of execution involves the ordering of each matrix in a multiplication:

$$\mathrm{Rot}(\gamma, v) = A_2 A_3 \cdot \mathrm{Rot}(z, v) \cdot A_3^T A_2^T$$

Minimum description of directions with Euler angle:

The formed Euler angle describes the minimum direction by combining the independent linear components of the rotation matrix in the current coordinate system (three rotations around three axes of three different reference systems). Depending on the specific combination of 3

independent components from the original 9 components, 12 different Euler angles can be obtained.

In the Euler rotation ZYZ = (φ,ϑ,ψ)
R_{EUL} = Rot(z, φ).Rot(y', ϑ).Rot(z", ψ)

Results of matrix multiplication:

$$R_{EUL} = \mathrm{Rot}(z,\varphi)\cdot\mathrm{Rot}(y,\vartheta)\cdot\mathrm{Rot}(z'',\psi)$$
$$= \begin{bmatrix} c_\varphi c_\vartheta c_\psi - s_\varphi s_\psi & -c_\varphi c_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta \\ s_\varphi c_\vartheta c_\psi + c_\varphi s_\psi & -s_\varphi c_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta \\ -s_\vartheta c_\psi & s_\vartheta s_\psi & c_\vartheta \end{bmatrix}$$

Given a matrix after multiplying by specific angles:

$$R = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The last column of the two matrices has the simplest form, so we can create the following system of equations:

$$\begin{cases} c_\varphi s_\vartheta = a_{13} \\ s_\varphi s_\vartheta = a_{23} \\ c_\vartheta = a_{33} \end{cases}$$

Roll - pitch - Yaw angle:
People often liken this to the oscillation of a ship. Basing on what Euler angle is, we can briefly understand that RPY is just a set of EULER angular sets (zyx) = (but the difference is basically that three rotations are done around three axes of the same original frame of reference.

$$R_{RPY} = R(z,\varphi)R(y,\vartheta)R(x,\psi)$$
$$= \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi S_\vartheta S_\psi - S_\varphi c_\psi & c_\varphi S_\vartheta c_\psi + S_\varphi S_\psi \\ S_\varphi c_\vartheta & S_\varphi S_\vartheta S_\psi + c_\varphi c_\psi & S_\varphi S_\vartheta c_\psi - c_\varphi S_\psi \\ -S_\vartheta & c_\vartheta S_\psi & c_\vartheta c_\psi \end{bmatrix}$$

Similar to the case of EULER angle, the inverse problem is solved by comparing the aforementioned result matrix with a given orientation matrix:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

The set of rotation parameters can be determined by homogenizing the corresponding elements creating a system of three hidden triple equations. In general, Euler rotation and RPY rotation differ in that:

- The rotation rotation three times around three axes of three different frames of reference, the position of the object to be located with the frame of rotation turned three times is determined by the Euler matrix. It is essentially an object that achieves its orientation by rotating the frame of reference and remains standing by itself. (The object turned away in this rotation is the frame of reference)
- RPY rotation re-locates the object by keeping the reference frame fixed while rotating objects consecutively three times around the three axes of the original frame of reference. (The object that rotates in this rotation is the object.)
- Rotation of a continuous reference frame (Euler) along the axes of the new generated local reference frame (under fixed object conditions) gives the same result as continuous object rotation (RPY) compared to the reference system Projector projection in parallel in reverse order.
  (Mark W. Spong, 2005)

2.5.2.3. Kinetics of the mechanical arm

Kinetic is that when given the joint parameters to determine the position and orientation of all joints and links on the arm, usually if not control the trajectory of the arm to avoid collision with the object. Unlike in the work area, people often only specify the position and orientation of the last joint (end effector). Manipulators are formed from combinations of links and joints. The manipulator has two basic forms, which can be dynamically formed so that it is closed or open. The links and joints are described through parameters divided into two types, the parameters do not change (link length) called parameters. The parameters that change (the rotation angle of links and joints, the amount of long movement of the translational joints) are called the joint variable.

Coordinate conversion is represented by the homogeneous conversion matrix:

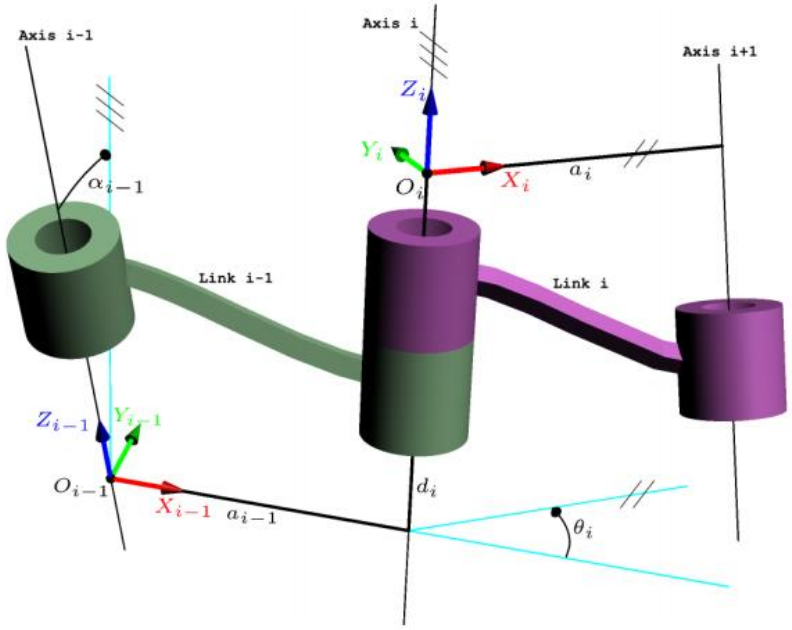
$$T^0(q) = \begin{bmatrix} n^0(q) & s^0(q) & a^0(q) & p^0(q) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where $p^0(q)$ is the positioning vector, $n^0(q), s^0(q), a^0(q)$ is the directional vector in the form of a cosine indicating the direction of the end effector. For instance, with a homogeneous matrix, we can choose as follows:

$$T^0(q) = \begin{bmatrix} - & a_{12} & a_{13} & a_{14} \\ - & - & a_{23} & a_{24} \\ - & - & - & a_{34} \\ - & - & - & 1 \end{bmatrix}$$

The elements a_{12}; a_{13}; a_{23} are oriented elements, a_{14} elements; a_{24}; a_{34} are positioning elements. So only six elements are needed to describe positioning and orientation.
In order to determine position and orientation of each link on the arm as well as the end effector, one must attach the extrapolation coordinate systems on each link, the whole structure has a common reference system connected to a fixed frame, this reference system is both to describe the position, orientation of the end effector of the machine arm, and to describe the impact object of the machine arm that it needs to identify. The construction of these reference systems needs to be highly uniform, requiring uniqueness. The following discussion is about DH rules, which is considered as a method for defining the position and orientation of joints and end effector of manipulator.

DH - Denavit–Hartenberg Convention:



In general, the machine arm is considered to have n joints, where the joint number I connects links (i) with links (i + 1) as shown. According to the DH rules, the coordinate systems are determined by the following convention:

- The coordinate axis zi coincides with the rotation axis of the joint (i+1), the root coincides with the foot of the common perpendicular line between the axis of rotation joint (i) and the axis of rotation joint (i+1), its x axis coincides with the common perpendicular line and direction from the axis (i-1) to the axis (i), the y axis itself determines according to the right hand rule.
- The coordinate axis z_{i-1} coincides with the rotation of the joint (i), the x-axis coincides with the common perpendicular line between the axis (i-1) and the joint (i), the positive direction from the axis (i-1) to the joint (i). The y-axis can be determined according to the right-hand rule.

Transform so that the Oi-1 reference frame coincides with the Oi reference frame. The transformation sequence is as follows: Translating Oi-1 along the axis (Oi-1zi-1) an amount equal to the translation matrix. After receiving the O'i frame of reference, rotating an angle around the z'i axis with the rotation matrix. Multiplying these matrices together has a homogeneous transformation matrix:

$$A_{i'}^{i-1} = \begin{bmatrix} c_{20} & s_{20} & 0 & 0 \\ s_{20} & c_{20} & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translating the O'i frame of reference to the x'i axis is equal to the translational matrix. Turn the frame of reference received in the upper step around the x'i axis to complete. Multiplying these matrices together has the homogeneous transformation matrix of this step as follows:

$$A_i^{i'} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c_\alpha & -s_\alpha & 0 \\ 0 & s_\alpha & c_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix of transformations obtained by multiplying the two matrices is as follows:

$$A_i^{i-1}(q_i) = A_{i'}^{i-1} A_i^{i'} = \begin{bmatrix} c_{\vartheta i} & -s_{\vartheta i} c_{\alpha i} & s_{\vartheta i} s_{\alpha i} & a_i c_{\vartheta i} \\ S_{\vartheta i} & c_{\vartheta i} c_{\alpha i} & -c_{\vartheta i} S_{\alpha i} & a_i S_{\vartheta i} \\ 0 & S_{\alpha i} & c_{\alpha i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

There are some special cases of DH rules. Firstly, the reference systems are located based on the intersection of the common perpendicular lines between the two rotating axes, so in the case of two parallel rotating axes, it is possible to choose the position of the reference system at its original position of reference system. Also, in that case the rotation around the x-axis is not necessary. Secondly, in the case where the two rotational axes intersect, the translational amount along the x-axis is zero (Mark W. Spong, 2005)

2.5.2.4. Inverse kinetic

The kinetic problem is to determine the position and orientation of the task when given the joint parameters. The inverse problem for the given position and orientation of the final operation requires the determination of a set of extrapolating coordinate parameters to ensure a given movement of the task. For open-chain kinematic structured manipulators, if the parameter set is given, the position and orientation of the manipulators are uniquely defined. This is not true for machines with closed-chain kinematic.

Derived from the basic dynamic equation we have:

$$T_n = A_1 \cdot A_2 \dots A_n = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The two matrices on both sides of the equation are homogeneous 4x4 matrices. Comparing the corresponding elements of the two matrices, we have six independent equations with qi, (i = 1 ... n). There are 3 possible cases: If (usually also the number of degrees of freedom of the robotic device) n <6, the robot mechanism only moves the end effector to limited positions and directions without bringing the end effector to any position and orientation. This case is applicable when there is no requirement to change some positioning parameters and the orientation of the end effector. If n = 6, the match set q1 ... q6 is completely determined. However, the solution is not always easy to find. Because generally the system of equations found is often transcendent and the solution of this system of equations does not always converge. If n> 6, there are likely to be many solutions, that is, reaching the same position and orientation of the end effector may have multiple sets of parameter variables qi.

Method 1: We have $T_n=T_i.\ {}^iT_n$, where ${}^iT_n$ is the conversion matrix from the i coordinate system to the n coordinate system. Multiplying the two sides by T_i^{-1} gives: $T_i^{-1}T_n= {}^iT_n$ and because $T_i^{-1}=(A_1.A_2...A_i)^{-1} =A_i^{-1}... A_2^{-1}.A_1^{-1}$ so that $A_i^{-1}... A_2^{-1}.A_1^{-1}T_n= {}^iT_n$. Combining them, we have

$$iT_n = A_i^{-1} \dots A_2^{-1} \cdot A_1^{-1} \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For each value of i, when comparing the corresponding elements of two matrices on both sides of the above expression, we have 6 independent equations for determining the matching variable qi (i = 1..n) .

Method 2: From the basic system of dynamic equations, we have six corresponding independent equations to determine the joint variable qi (i = 1..n). The independent equations:

$$\begin{gathered} f_1(q_1, q_2 \dots q_n) - p_x = 0 \\ \{f_2(q_1, q_2 \dots q_n) - p_y = 0 \\ f_3(q_1, q_2 \dots q_n) - p_z = 0 \end{gathered}$$

Depending on the structure of the robot, we have the remaining three equations, called the following equations:

$$\begin{gathered} f_4(q_1, q_2 \dots q_n) = 0 \\ \{f_5(q_1, q_2 \dots q_n) = 0 \\ f_6(q_1, q_2 \dots q_n) = 0 \end{gathered}$$

Solving the reverse kinematic problem of the robot structure is very complicated, so far there is no general algorithm for all cases. To solve these problems, people often take advantage of some properties of a structure to simplify the calculation and only find solutions for specific models, such as geometric characteristics, constraints of joint variables. Solving by conventional analytical methods is generally very difficult. Sometimes it can be solve, to overcome we use numerical methods to solve reverse robot kinematic problems. (Mark W. Spong, 2005)

2.6 **ROS**

To develop a 3D scanning system, like the 3D scanning system in this project, a series of sensors were used, including the LiDAR 2D sensor, color camera with RGB-D camera depth, Inertial Measurement Units, robot arm UR5, etc. And these devices are from different companies resulting in a lack of uniformity in structure and usage. In addition, processing data from these devices also requires the use of complex algorithms such as ELK filter, PID tunning, Kalman algorithm, etc. If every individual or organization wants to build their own scanning system, they have to rebuild the same algorithm, and the libraries to retrieve data from the sensor will cause a waste of time and effort. So, the need for robotic technology is that a system can eliminate these unnecessary costs, so researchers around the world can better contribute and solve difficult problems caused by robotic system. Developers need protocols to transfer data from one part of the system to another, they need unified tools and practices to build their own software and they need pre-written libraries to avoid it compatibility issues.

From the authors of the paper describing ROS:
“To meet these challenges, many robotics researchers, including ourselves, have previously created a wide variety of frameworks to manage complexity and facilitate rapid prototyping of software for experiments, resulting in themany robotic software systems currently used in academia and industry [1]. Each of these frameworks was designed for a particular purpose, perhaps in response to perceived weaknesses of other available frameworks, or to place emphasis on aspects which were seen as most important in the designprocess.” (Quigley, 2009)

And so, ROS was developed, a system that provides programmers with standard rules and ways to organize their tools so that they can collaborate on a large scale and have a consistency in every things. Robot Operation System (ROS) is an open source platform for designing and building robot software applications, providing a distributed control system on heterogeneous computer clusters. It was born with the purpose of allowing researchers to accelerate the development of new robotic systems, supporting code reuse through standard tools and

interfaces. ROS also features that make it an extremely suitable operating system for robots such as:
- Support asynchronous programming due to call-back programming
- The operating system is distributed so the processes are separate but can still connect through messages.
- Avoiding hardware dependencies due to the use of a messaging method.
- Operate and manage the system in a simple way through node lists

Because of these advantages of ROS, it has a wide community of users and developers. ROS provides an extensive library foundations and abundant, widely used and compatibility support with a lot of products from a lot of different companies.

Structurally, the ROS operating system consists of three levels: Filesystem, Computation Graph, and Community. The levels are closely linked and each level contains definitions and concepts about the components that make up the architecture and how the system works. Details of these levels will be described shortly.

ROS Filesystem level

According to the definitions of ROS, Filesystem is a ROS resource stored on system physical memory, including components such as:

- Data package is the main unit in the software structure of ROS operating system. A package may contain ROS execution commands (nodes), a library for ROS, data sets, configuration files, or other necessary data in the system. Package is the smallest atomic component built and put into use in ROS. This means that the smallest component we can build and put into use in ROS is a package.
- Metapackages: These are specific packages that only group other related packages. Most metapackages are used in the same way as compatibility later when converting to the rosbuild stack.
- Package Manifests: a package data manifest (package.xml), which provides metadata about the package including the name, version, license information and package dependencies. whether that. The Manifest also contains information about programming language features such as the compiler flags
- Stacks: is a collection of packages that works together to perform a specific function, for instance a

"navigation stack" which is a collection of instructions packages for robots.

- Stack Manifests: (stack.xml) provides information about a stack, including license information and parameters depending on other stacks.
- Message (msg) Types: Message description information, stored in my_package / msg / MyMessageType.msg, defines the data structure for messages sent in ROS.
- Service (srv) Types: information describing services, stored in my_package / srv / MyServiceType.srv, defining the data structure for access commands (responses) and responses (responses) of services in ROS.

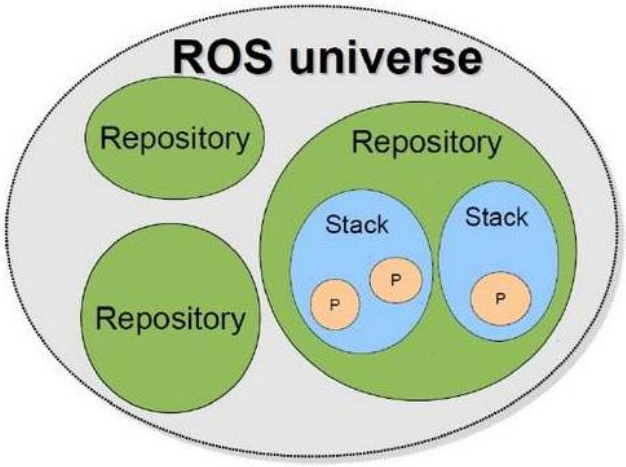


Figure 21. Ros universe and repository

Computation Graph layer

The "Computation Graph" (computational scheme) is a peer-to-peer network of ROS processes that perform processing together. The basic Graph Computation consists of elements: nodes, Master, Parameter Server, messages, servicer, topics, and bags. All of these components provide data to the Graph in different ways

- Nodes: are processes that perform the calculation. ROS operating system is modularized, a robot control system usually contains many nodes, nodes must be built in detail and specialized. For example, a node controlling the sensor system, a node controlling the wheel motor, a node performing positioning tasks, a node planning a path, a node drawing orbit of the

system. ROS client library, such as roscpp or rospy, is used to build the nodes of ROS.

- Master: The ROS Master provides the registered name and lookup to the rest of the Computation Graph. In general, the Master is like a central system that manages the address and data flow exchanged between Nodes. Without the Master, nodes would not be able to find each other, exchange data, or perform services and actions
- Messages: Message is designed as a means and is a general definition for communication between nodes for the purpose of standardizing communication. A simple message is a data structure, consisting of typed fields. Standard data types (such as integers, floating points, booleans) and arrays (arrays) with standard types can be integrated into Messages. Messages can include nested structures and arrays (like the structs in the C language).
- Topics: Messages are routed through an exchange system (transport system), which categorizes messages into two forms: publish and subscribe (subscribe to receive information). A node sends a message by giving information to a topic. The topic name is used to specify the content of the message. A node related to the type of data will subscribe to the corresponding topic. A topic may have many news subscribers (publishers) as well as multiple subscribers (subcribers); and each node can also broadcast a variety of topics, as well as receive messages from multiple topics. The sources of communication and the recipients generally do not need to know about each other's existence. Thereby we see the structure of Topic in ROS is separating the information source from the information user. Topics are considered to be a channel of styled messages. Each channel has a unique name, and any node can connect to this channel to send / receive messages, as long as the message is the same type as that topic.

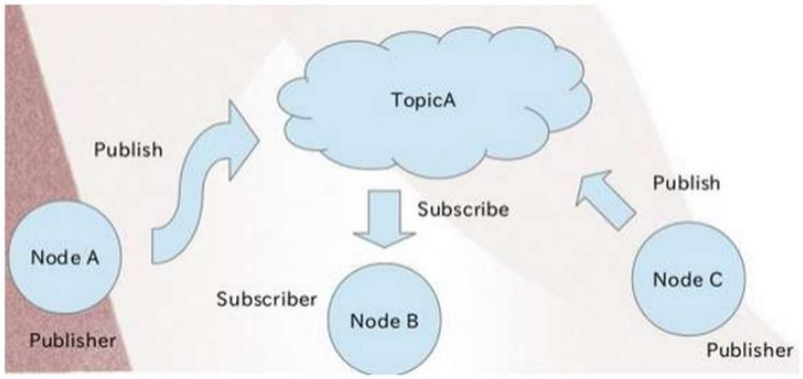


Figure 22. Relationship between action and topic

- Services: The model of publish / subcribe as described above is a very flexible model, however, its characteristics are information that is multi-object, one-way transmission (many-to-many, one -way) is sometimes not suitable for situations that require interaction in the form of request / reply (request / response), this type of interaction is often found in distribution systems. Therefore, there should be one more component in ROS Graph, which is "services", in order to make requests for interaction in the form of request / reply. Services are a pair of message structures: a message to send the request and a message to respond. A node provides a service with a "name" attribute, a client uses that service by sending a request message and waiting for a response. In the ROS client library, this interaction method is often provided as a function called remotely.
- Bags: is a format to store and play back data from ROS messages. Bags are an essential and important feature for storing data, such as sensor data, which is needed to develop and test algorithms.
- ROS Master acts as a nameservice in the entire Computation Graph level structure in ROS. The Master is basically designed to store registration information for topics and services in an effort to establish a network for nodes. Each node's registration information is required during initialization, and it must be sent to the Master by the node itself. When communicating with the Master, nodes can also receive information about other nodes registered with the Master from which to make the appropriate

connection to exchange data. The master will also send callbacks to the nodes in the event of a change in the registration information with the Master, allowing nodes to automatically make a connection when new nodes join the network.

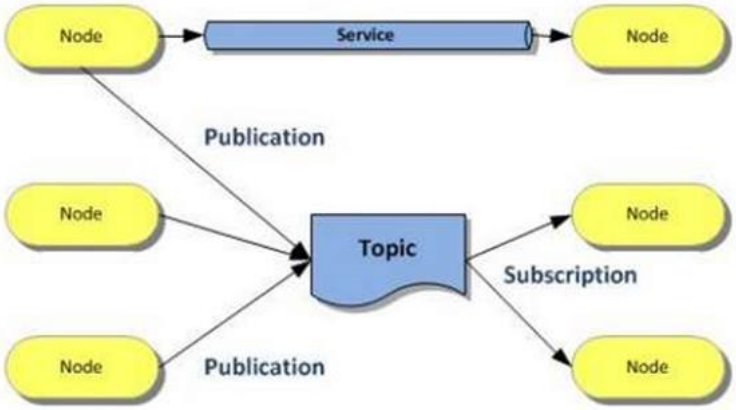


Figure 23. Peer-to-peer messages

The construction of the system as in ROS allows information providers and recipients to be separated, and the relationship is made through the "name" attribute. "Name" is a property that plays a very important role. Important in ROS: the names of nodes, topics, services, and parameters are mandatory and must be unique. Moreover, every ROS Client library supports the command-line to link these names (remapping names), so that the compiled program can then be reconfigured even when operating in other Graph Computation.

Community level

The concept of connection layer of ROS is understood as the ROS resources that the user community can exchange with each other. It could be software or related knowledge. These types of resources include:

- Distribute: Distribute: a list of distributed ROS versions that, depending on the characteristics of the main operating system, will be installed appropriately by the user.
- Repositories: The strength of ROS is that operations can benefit from relying on an interconnected network of code repositories, where different organizations can develop and publish their own software components. This will contribute to increasing the community in the construction of the ROS ecosystem.

- Other components such as the ROS Wiki act as a forum to provide official documentation and instructions to users, etc.

3 SYSTEM OVERVIEW

3.1 RGBD camera – realsense D435i

Figure 24. Intel RealSense Depth Camera D435i

In this project, we use RGB-D realsense D435i camera from Intel company. The "D" in RGB-D stands for Depth, which means realsense D435i's output is the color depth image data stream. This camera is integrated with many technologies, which are considered the main core in this project. Realsense 435i eliver 1280- × 720-pixel streams at 30 frames/s or a lower-resolution 848- × 480-pixel stream at 90 frames/s integrated with inertial measurement unit (IMU) to keep track on movement and rotation of the camera in 6 degrees of freedom (6DoF). This sensor plays an important role in image stabilization. Basic specifications of Realsense D435i can be found in the following table.

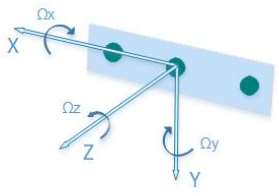
TECH SPECS

Features	Use Environment: Indoor/Outdoor Image Sensor Technology: Global Shutter, 3µm x 3µm pixel size	Maximum Range: Approx. 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
Depth	Depth Technology: Active IR Stereo Depth Field of View (FOV): 87°±3° x 58°±1° x 95°±3°	Minimum Depth Distance (Min-Z): 0.105 m Depth Output Resolution & Frame Rate: Up to 1280 x 720 active stereo depth resolution. Up to 90 fps.
RGB	RGB Sensor Resolution & Frame Rate): 1920 x 1080 RGB Frame Rate: 30 fps	RGB Sensor FOV (H x V x D): 69.4° x 42.5° x 77° (+/- 3°)
Major Components	Camera Module: Intel RealSense Module D430 + RGB Camera	Vision Processor Board: Intel RealSense Vision Processor D4
Physical	Form Factor: Camera Peripheral Length x Depth x Height: 90 mm x 25 mm x 25 mm	Connectors: USB-C® 3.1 Gen 1* Mounting Mechanism: One 1/4-20 UNC thread mounting point. Two M3 thread mounting points.

Figure 25. Basic specifications of camera Realsense D435i

IMU - Inertial measurement unit

IMU has become one of the essential sensors in the robotic industry. From drone-controlled drones to civil aviation aircraft using IMU sensors range from simple to complex to detect the movement direction of the system, such as roll, pitch, yaw, accelerometer, etc.



1. The positive x-axis points to the right.
2. The positive y-axis points down.
3. The positive z-axis points forward

Figure 26. Resulted orientation angles and acceleration vectors share the coordinate system with the depth sensor.

An IMU module will consist of two sensors: the acceleration sensor (accelerometer) and rotary sensor (gyroscope). Accelerometer (abbreviated as Accel): as the name implies, Accel is simply an accelerometer sensor and usually has three xyz axes corresponding to three dimensions of space (one or two axis is less commonly used). Note that Accel measures the acceleration of gravity, so the actual value will include gravity. Gyroscope (referred to as gyro): is a type of sensor that measures its rotation speed around an axis. Similar to Accel, gyro usually

has 3 xyz axes. A simple example, when putting an IMU chip upright as figure 26 and leave it motionless, the return value will be accel = [0.0, -9.8, 0.0] and gyro = [0.0, 0.0, 0.0] because only the gravity of the earth exerts force and does not have any rotation. Note that the gyro only measures the rotation speed, not the rotation angle directly, so when rotating the module a certain angle and then stopping, the value of the gyro will increase and then lower to 0. A full IMU module will be called 6-DOF (6 Degrees Of Freedom) meaning6 independent axes (3 of Accel and 3 of gyro). Although sometimes this is not enough, more complex projects such as navigating aircraft or robots may require 9-DOF (adding a 3-axis magnetic field sensor - magnetometer - that works almost like a compass for orientation), or 10-DOF (add a barometer - used to measure altitude) or even 11-DOF (add GPS module to locate).

The IMU built into Realsense D435i is the Bosch BMI055 sensor with six degrees of freedom (6DOF) including a 3D 12-bit acceleration sensor and a 16-bit, ±2000°/s gyroscope. The BMI055 is designed to deliver low-noise measurements of angular rates and acceleration. However, one note when using the IMU sensor built into the realsense camera is the built-in IMU can only keep track for a very short time. Moving or turning too quickly will break the sequence of successful point cloud matches and will result in the system losing track. It could happen that the system will recover immediately if stopped moving but if not, the longer the time passed since the break, the farther away it will drift from the correct position.

3.2 **LiDAR**

In this project, LiDAR sensor – TiM561, from SICK company is the sensor used mainly for the purpose of system positioning. The TiM561 sensor is not merely object detection device but also a non-contact ranging method within the Tim series. One of its modern technology is HDDM which stands for high definition distance measurement, allowing TiM561 to monitor large space for both inside and outside applications without concern about the surface or the amount of ambient light. TiM5xx is an effective solution for accurate measurement data from the scanned surface, allowing additional information such as the size and shape of an object with a slight margin of error.

Figure 27. LiDAR sensor – TiM561

3.3 UR5

The UR is the ideal robot arm for automating tasks that handle lightweight products such as lifting, placing and checking. UR5 is one of three robot arm products developed by Universal Robot company. The only robot arm products that are currently commercialized by this Danish company are UR3, UR5, UR10. The advantages of these products are optimization of investment, fast installation, flexible deployment and safety. The robot body has a quite compact, lightweight, space-saving design and structure with 6-axis arms designed minimalist, rounded, to operate safely with humans. In term of safety, the force sensors are integrated in the joints allowing it to automatically stop working when colliding with an unwanted object. A simple comparison between UR5 with another robot arm product of the same size and weight developed by ABB company – IRB1200, revealing that ABB product has a repeatability of 0.02 mm, which is lower than the published specification of UR5, 0.1 mm. However, of course to achieve such low repeatability, the products are in the same segment with UR5 must have a heavier and sturdier design, which also means that they are more dangerous. This results in the installation of protective cages around these robot arms being required. Therefore, by choosing UR5, the flexibility and safety are emphasized and enhanced in this project

3.4 Raspberry Pi

The Raspberry Pi is a compact computer, about the size of an ATM card. Features of the Raspberry Pi build around the SoC Broadcom BCM2835 processor - It is a powerful mobile processor chip of small size or used in

mobile phones, including CPU, GPU, audio/video processoretc. All are integrated inside this low power chip. Basically Raspberry Pi has the ability to support Linux OS because Pi's raspbian operating system is built on Linux but there is still the absence of Ubuntu (due to ARMv6 CPU). Some Linux distributions (embedded) running on Raspberry Pi such as Raspbian, Pidora, openSUSE, OpenWRT, OpenELEC.

The official name of the latest raspberry pi, which is also the version used in this project, is the Raspberry Pi 4. Older Raspberry Pi series such as Raspberry Pi 3B use only LPDDR2 RAM and have a capacity of 1GB, old 1.4-GHz quad-core ARM Cortex-A53 CPU and USB 2.0. The Raspberry Pi 4 could be upgraded to 2GB or 4GB of RAM, along with a USB 3.1 port to be able to connect to many other devices. This USB 3 port is very important in the project, because it allows data access with realsense cameras with transfer rates up to 4.8 - 5 Gbps, ie 600 - 625MB / s in theory, help optimize the scanning process is carried out accurately, close to real time and avoid data loss, drop frame.
The Raspberry Pi in this project was used to run ROS (Robot Operating System), which is an intermediary software with sophisticated message transfer protocols. ROS will be discussed in more detail later in this chapter.

3.5 **ROS dependencies/ library**

3.5.1 Realsense-ros

Realsense-ros is a package for using Realsense camera as D400 series SR300 camera and T265 Tracking Module with ROS. It was developed with compatibility with the Kinetic version on Ubuntu 16.04 and Melodic on Ubuntu 18.04 of ROS. As a complete and sophisticated version for extracting a variety of raw data formats from Realsense camera lines, this package offers a wide range of customizable parameters allowing users to configure the package operation as well as set the output according to the target use with various outputs from camera information, raw image, depth image, extracted data from IMU built-in camera etcHighly flexible and customizable, this package is considered to be an important first package in this project to extract data from Realsense D435i cameras. The data from this package will be used for a variety of purposes, but it is still primarily about providing input for RTABMAP to implement SLAM as a core task of the project.

3.5.2 Sick_scan

This is a library for TiM and MRS laser scanners from SICK sensor manufacturing company, built on the foundation of the sick_tim package. Package supports the acquisition of raw data from the laser sensor, its

output is data in the form of PointCloud2 and LaserScan. In addition, it can be expanded to support the acquisition of data from the IMU sensor of the MRS6xxx sensor line as well as the MRS1xxx sensor series in the future. Similar to Sick_scan, this library package provides a variety of user-adjustable parameters so that it can set up different configurations for different uses. In addition, it also provides some useful tools during use such as sick_generic_device_finder.py to find the scanner IP address of the entire network, or sick_new_ip.launch allowing users to change the scanner's IP address when using Ethernet for sensor data transfer without using Sick's Sopas software. In terms of compatibility with ROS versions, it is supported to run on two main distros, kinetic and melodic. This is the main ROS package used in this project to extract raw data from the LiDAR Sick TiM561 sensor.

3.5.3 RTABMAP

Basically, SLAM algorithm can be divided into 5 groups, listed as Extended Kalman Filter SLAM (EKF), Sparse Extended Information Filter (SEIF), Extended Information Form (EIF), FastSLAM, GraphSLAM. The two most useful approaches to SLAM are Grid based FastSLAM and GraphSLAM (Sebastian Thrun, May 2006). RTAB-Map is an RGB-D Graph SLAM library with a global Bayesianloop closure detector. The capacity required by this filter is calculated using quantization of visual words (SURF feature). If closed loop is accepted (enough inliers), the link is added to the chart. Converting between images is calculated using RANSAC. The TORO (Tree-based network optimizer) method is used to optimize the scheme (with the positions of nodes and transforms as constraints). Once the loop is found, the update can spread across the entire chart and fix the map. (Labbé & Michaud, Sept 2014)

For RTABMAP versions designed for use in ROS, also known as RTABMAP-ROS supportting the most of current versions of ROS from Hydro to Jade. The equipment that can be used with RTABMAP is quite diverse such as color camcorders with RGB-D depth sensor, or LiDAR laser sensors, etc. n addition, RTABMAP accepts external odometry sources provided by 3rd parties, however, if this external odometry source is not available, it is possible to use its own odometry source, which is extracted and calculated from the camera's input data to solve the egg and chicken problem of localization and mapping at the same time. However, this calculation is based substantially on the tracking feature on image, nên trong một số môi trường quét đặc thù với quá ít feature, việc sử dụng nguồn odometry từ bên ngoài ví dụ như encoder của bánh xe, hay LiDAR is needed for rtabmap to work properly. In addition, the customization capabilities of RTABMAP are high because it provides a variety of parameters that can be modified according to user needs.

Overall, rtabmap is a library used as the central ROS package of this project, other libraries such as imu_filter_madgwick,

laser_scan_matching are mostly used to revolve around the support for SLAM done by this library that goes smoothly and accurately.

3.5.4 Laser_scan_matcher

This is a library that performs laser scan matching incrementally, built on pure C implementation of ICP variant – The canimical Scan Matcher (CSM). CSM is an ICP variant using a point-to-line metric optimized for range-finder scan matching. The ability of this variant has been proven through practical use when it is now commonly used in mobile experimental robots in industry (CSM). The Laser_scan_matcher library operates with a high independence because unlike similar libraries, it does not require an odometry supply from the outside. In addition to the various inputs that can be made optional to the library, the main input for Laser_scan_matcher is LaserScan message sensor_msgs/LaserScan or sensor_msgs/PointCloud2 message, and the output is the location of the system in space as a Pose2D message in the geometry_msgs package or a tf transform (ROS wiki). However, because it is open source, it has the ability to leverage the underlying library base, combining source code editing so that the final output can be used as a source of geometry, used continue as an input for rtabmap in the setting using laser scan.

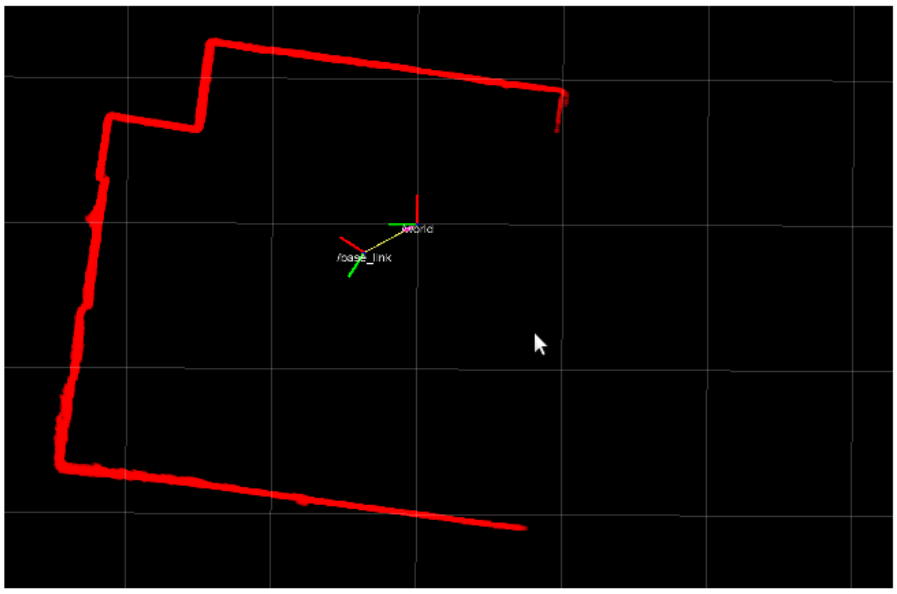


Figure 28. Output of package Laser_scan_matcher on Rviz

3.5.5 imu_filter_madgwick

The IMU is used as an important package to collect the system angular rate and orientation of the scanner, which is used not only to represent the system's posture but also to improve the process when implementing

the SLAM of RTABMAP. The imu_filter_madgwick package is used to filter and fuse raw data from IMU devices. It fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into an orientation quaternion, and publishes the fused data on the imu / data topic (imu_filter_madgwick). This package can be used with most IMU types, as long as the IMU provides angular velocity and linear acceleration data. The type of input data supported for this package in ROs environment is Message containing raw IMU data, including angular velocities and linear accelerations (sensor_msgs / imu) and Magnetic field vector (sensor_msgs / Magnetic Field). The fused Imu message, containing the orientation as a message sensor_msgs / Imu will be provided as an output for the package. (Ros Wiki)

3.5.6 Moveit

UR5, as presented, will act as a robotic arm for accurate, uniform movement of the scanner and with a programmed path planning, which can vary based on application, scanning target as well as the subject to be scanned. MoveIt provides functions for kinetics, motion planning, 3D awareness etc, and runs on top of ROS. It has a very strong system architecture, can handle scripts in Python or C ++, and it can easily communicate with RVIZ. In addition, it can naturally associate with ROS through most of the basic methods that the packages in ROS still use to communicate with each other, making it easy to interact with and integrate Moveit into any application or system.

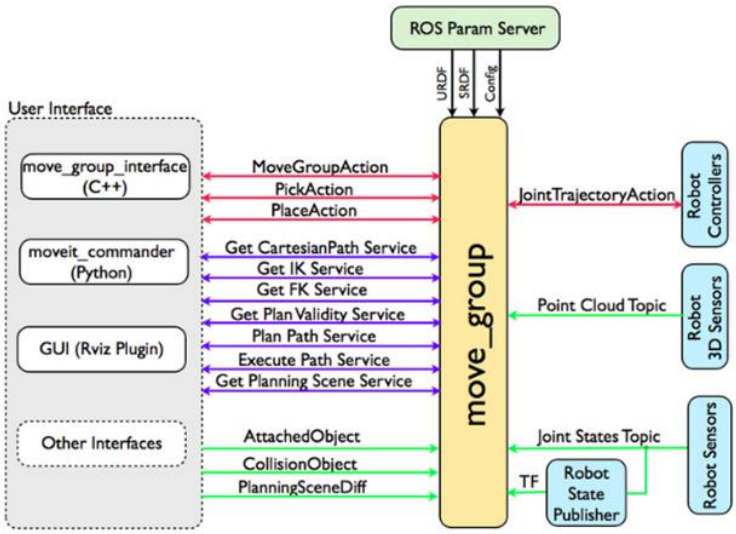


Figure 29. Moveit system architecture

Moveit provides a lot of tools, functions as well as interfaces so that users can ultilize this open source library in many different ways. In this project, the two main functions of the Moveit were used are the Python interface APIs, to implement programming to plan the trajectory of the robot arm

and RVIZ connectivity with Moveit to use path planner directly on the user-friendly interface, even for those who are not programming experts.

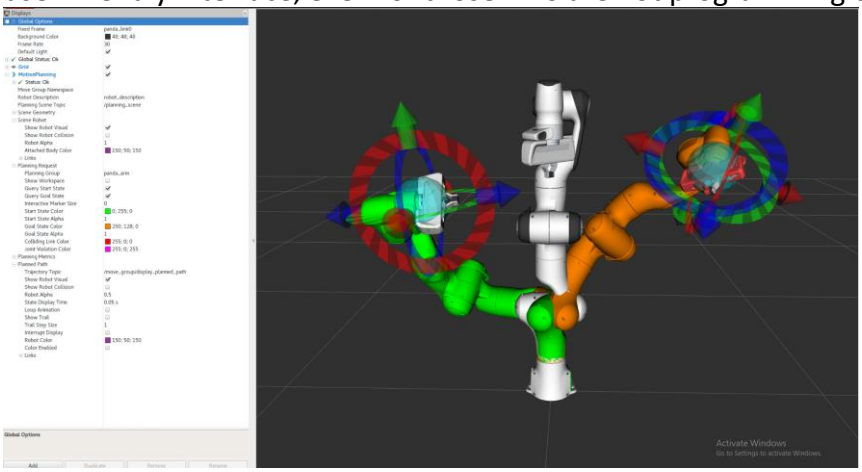


Figure 30. Path planner in Moveit with RVIZ

3.5.7 UR_modern_driver

Ur_modern is a driver for the UR3/UR5/UR10 robot arms from Universal Robots. It is designed to replace the old driver transparently (ur_robot_driver), while solving some issues, improving usability as well as enabling compatibility of ros_control. The ur_modern_do package was developed by Thomas Timm Andersen, a robotics engineer, as part of a doctoral thesis on sensor-based real-time robot control. He developed ur_modern_driver to improve some of the original driver deficiencies that the robot user community has been struggling for years.

The driver provides two interfaces; a joint velocity interface and a joint trajectory interface. The tasks performed by the velocity interface are performed at a higher speed because speed data is collected and sent to the robot controller closer to real time. Using the velocity interface allows programmed tasks to be performed with lower latency and simpler. The orbital interface is the interface of action, getting the goals and verifying that the trajectory is valid. If the trajectory is good, the driver performs a tertiary interpolation between orbital targets and transmits intermittent position matching to a custom script running on the controller.

Although largely omitted, the custom scripts used in this package are based on the scripts used in the original ur_do. The package implements a constant stream of communication to read data from the socket and update the global variable with the latest goals, and the control flow reads global variables and updates new setpoints for the controller. That's why ur_modern_driver can send new locations much faster than the frequency of the controller.

The Raspberry Pi is used for the purpose of operating the ur_modern_driver package installed with the ROS package Kinetic

version, which allows synchronization of message structures with ROS Kinetic devices running on the same system. The Raspberry Pi will communicate with the robot via the Ethernet connection port, while communicating with other ROS-powered devices in the system via WLAN communication protocols.

3.6 **Summary**

All of these hardware devices are selected based on the low cost criteria but still meet the quality requirement, At the same time, considering compatibility with ROS as well as in accordance with the future development plan, thereby building a homogeneous system, improving the autonomy for the system, openning up many possibilities for upgrading and comprehensive automation for the system.
However, these devices are not irreplaceable. They are all customizable to replace by other devices with a structure and relative characteristics depending on the changing goals of the project or the user's expanding needs.

4 **IMPLEMENTATION**

The main objective of this chapter is to focus on explaining the different settings, configurations of camera, sensor and hardware structure for different uses but aim to achieve the best possible quality results. After testing different approaches, settings and comparing their 3D mesh generated, the optimal options for the system have been selected. These settings can be basically divided into two main approaches as developing scanning system as a handheld scanner and basing on handheld approach structure, adjustments was made to set up a scanning system movement scheduled by robot arm or any linear sliding support systems. Note that, with the hardware structure as well as the handheld scanner's software settings, it can be attached to a fixed position if the surface of object needed to be scanned within the camera's FOV. With each approach, there are different hardware and software configurations to increase the compatibility of the system with different scanning targets and operating environments. The settings of each approach and their different configurations will be detailed in the System architecture section below.

But first, it is necessary to mention how to transmit data from sensors, cameras to raw image data processing devices, and from control devices such as computers that send commands to devices other devices such as Raspberry Pi or UR5 to execute commands as required. This project was originally designed with the goal of developing a remotely controlled system. Therefore, this project establishes a local area network for sensor data transmission as well as transmitting command from the host

computer to robot and devices. This approach helps overcome the limitations of technology at the present time such as:

- Limited length of large data transmission cables.
- Homogeneity in IP addresses management of devices and sensors.
- The fixation of a computer to operate system etc

The benefits of a remotely controllable system are quite practical and convincing. When the system is put into use, one or more computers set up to use can participate in the process of the system rather than a single computer connected to a single output data cable of sensor. This means that users can be anywhere, possibly in the control room and not necessarily a specific computer or a fixed computer as long as the computers and devices are connected in the network. Thereby it is possible to order remote systems without having to be present at the field. Although this method has a weakness can be improved is the latency in data transmission. Data flows and commands in the network are, sometime, quite large leading to data latency and sometimes data loss, more or less affect the quality of scanning process. This can sometimes happen with different scanning methods as well as their different expected output, for instance the amount of data to be processed when it is necessary to scan the entire concrete block will be much higher than the mount of data when scanning a cross section of a concrete slab. However this problem could be improved. However, this problem can be minimized to an extent that its benefits outweigh the technological limitations it brings.

4.1 **System environment**

Because the ROS packages used in this project are designed to support installation and operation on ROS Kinetic, both ROS-equipped devices are Raspberry Pi and image data processing computers are installed with the Kinetic version of ROS. ROS will be installed on Ubuntu 16.04 LTS platform for image processing computers and the latest raspbian version, Buster for Raspberry Pi. When implementing the system some compatibility problems needed to be solved

- Realsense D435i needs to be connected to USB 3.0 port because of its large image data stream. However, not all USB 3.0 controllers are compatible. The image processing computer has only ASMedia USB 3.0 controllers, basically not working with Realsense camera as well as the camera's accompanying support library. Therefore, connecting the camera directly to the computer is not feasible, not to mention that the USB 3.0 cord will be very short, greatly affecting the mobility of the system with the type of camera connection directly to the processing image device.

Therefore, the use of small Raspberry Pi to collect data from the camera is a solution that is not only cheap and reliable but also considerably increases the mobility of the system.

- Installing realsense2_camera packages for Realsense D435i, Sick_scan for LiDAR LM651 or moveit for UR5 arm robots on Raspberry Pi is quite difficult because the packages themselves and ROS Kinetic in particular and other ROS distributions are not supported for official installation on Raspbian operating system. Installing packages, including the installation of ROS Kinetic on the Raspberry Pi, requires manual installation from the source, which is not feasible to install using the command line like "apt-get install <package_name>".

4.2 **General system architecture**

The scanning system in this thesis is a low-cost built prototype capable of mapping and localizing using RGBD-cameras as the main factor, in addition to having a 2D LiDAR sensor. Designed on the remote control and mapping criteria through data transmission in the network (TCP / IP). The general procedure for this system to perform 3D reconstruction will include the following:

1. Raspberry Pi model 4: ROS is installed on Raspberry pi to run realsense-ros package for the purpose of extracting image data or IMU data, sending this image data through local area network to Master, here is an image processing device and can also be used to perform other tasks such as synchronizing or compressing raw image data before sending to the Master to reduce data loss or asynchronous in data. In other settings that use the LiDAR or the UR5 arm arm, the Raspberry Pi is used to collect Laser scan data from LiDAR similar to the case of rgb-d cameras and is used as an implementation and transmission device to send control commands of the robot arm from the Moveit package run on the ROS platform are installed directly on the Raspberry Pi itself.
2. RGB-D camera - Realsense D435i: A color camera (RGB) with a depth sensor (Depth). Important data collected from the realsense2_camera package running in ROS installed on the Raspberry Pi can be listed such as movement and rotation in 6 DoF data from sensors and gyroscopes, raw image data aligned depth to color, data raw color images rgb etc. These data are directly input to ROS packages that implement SLAM such as RTABMAP or indirectly provided data for

imu_filter_madgwick package whose output is fused imu data provided back to RTABMAP.

3. Imu_filter_madgwick: The data from IMU of realsense camera is very important in this project, the data provided by IMU will help to improve the quality of SLAM better. The ROS package Imu_filter_madgwick package of ROS will be used to fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into an orientation quaternion, and publishes the fused data in the standard form. Input of the package feed by IMU data output of realsense2_camera through topics.

4. RTABMAP: As mentioned before, RTABMAP is the package used to implement SLAM in the project, with the main data are rgb color image and depth image provided from the realsense2_camera package. In addition, RTABMAP accepts IMU data that has sensor_msgs / Imu standard message structure from external sources, such as IMU message output of imu_filter_magdwick package to improve mapping quality. With the settings using only rgbd-d camera for SLAM, the odometry used for RTABMAP will be the source of visual odometry created by the package itself during mapping using algorithms calculated from depth image data as well as image features in the stream of image data collected from the provided environment. However, it is possible to use external odometry sources, with the setup using the LiDAR sensor, laser scan data via the 3rd package (Laser_scan_matcher) in ROS can be processed into external odometry sources provided for rtabmap. The visual information from the environment is processed and provided output through published topics such as cloud_map (Mapping 3D point cloud generated with the 3D point clouds), scan_map (Mapping 3D point cloud generated with the 2D scans or 3D scans) etc. Mesh file created from point_cloud during scanning of concrete beams in .ply format is also one type of output of RTABMAP, this file will be used to measure the actual size of an object using a variety of methods.

5. Data transmission between Raspberry Pi and computer image processing / Remote control and SLAM:
   Systems consist of many devices that require communication between processors running on different hardware. Depending on the data exchanged between devices, different communication methods may be used.
   - Option 1: Create a Python server on one device on the network, the other devices act as clients. Contact between these server and client pairs is done on the python interface. Files requested from another device can be transferred through this setup system.
   - Option 2: Establish ROS communication between systems with one computer running the ROS master

and other computers connecting to the ROS master via the same local network.

However, for a Python data transfer system, it requires the exchange of files and thus, files are created and removed every time data is transferred. This causes a greater delay than the method of transmitting message data through ROS master, which in turn leads to less efficiency. Therefore, communication via ROS master is a better option

The key to performing tasks with remote ROS is the ability to work on the same ROS master of many different devices. In other words, ROS implements communications between packages with nodes and topics on a ROS master. If all devices share the same ROS master, it means that multiple devices can interact with each other through certain nodes and topics on the Master. ROS is designed with distributed computing. A well-written node makes no assumptions about where it runs in the network, allowing computations to be moved at runtime to match the available resources (for example, having a driver node communicating with a hardware must run on a machine on which it is physically connected). So deploying a ROS system on many machines is simple (ROS master). With the ability to set up IP configuration easily on raspberry pi, configuring a network of devices according to the publisher-listener principle that shares the same ROS master running on image processing equipment on the network is possible with any device. Every device inside local network can join to publish or subscribe to data from these topics.

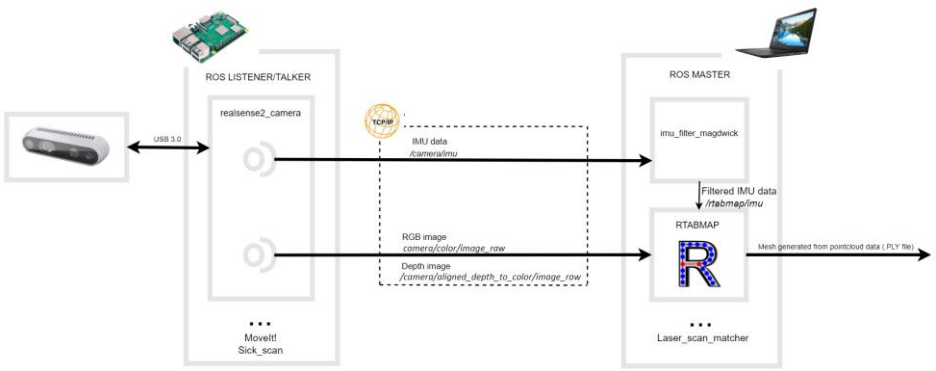


Figure 31. General structure of the system

Raspberry Pi is a device that collects image data stream from realsense camera, and with internet connectivity, it sends this data via network to computer for image processing. ROS master is installed on image processing device. ROS core on Raspberry Pi connects to ROS master to carry out data transfer. Both the Raspberry Pi and the image processing device are installed to perform data acquisition as well as mapping from collected image data. On the Raspberry Pi is the realsense2_camera package and on the image processing computer is the imu_filter_magdwick package to process data from IMU along RTABMAP to handle SLAM. In summary, this is a common system structure for the entire system and handheld scanner uses this setup completely. Although

some use the LiDAR 2D, arm UR5 robot to improve some inherent limitations of the system encountering, it requires some minor changes in hardware design as well as software settings. They are completely expanded based on this setting or only minor adjustments are made.

4.3 **Handheld setup**

Most scanners currently on the market are large in size, some are even connected to an even larger structure, which limits the number of objects that are suitable for scanning with small sizes, usually smaller than scanning equipment. For systems scanned around a table area or rotation table, the size and weight of the object to be scanned must match that table. The use of mobile scanning tools can increase the number of objects that can be scanned, thereby increasing the practical applicability of the device. Basically, handheld is an approach that uses an established system that has a structure almost similar to the general architecture just outlined above. In this section, details on system setup and hardware design were discussed

4.3.1 Software structure of system

4.3.1.1. Realsense2_camera

For further clarification, realsense D435i will be connected to Raspberry Pi via USB 3.0 with data transfer rate of up to 5 Gbps / 620 MBps in theory (Raspberry Pi 4 specs and benchmarks). With a high data rate supported from USB 3.0, the image data stream has a minimum frame rate of 6Hz to a maximum of 90Hz from the camera with resolution supported for the maximum RGB color picture frame up to 1920x1080, frame depth of 1280x720 which can be collected through the use of the realsense2_camera package running on ROS installed on the Raspberry Pi. However, choosing the resolution for the output image data also needs to consider the suitability of the input of the data to be sent, the input of RTABMAP. In theory, RTABMAP can handle all color and depth image data pairs as long as the ratio of the width divided by the height of the pair of images is equal. For example, if the resolution of the image depth is 1280x720, this ratio is approximately 1.7777 and equal to ratio of the width to the height of the rgb color resolution 1920x1080. So, in theory, RTABMAP works well with the resolution of the input image with 1280x720 for depth images and 1920x1080 for color images. However, during RTABMAP's actual SLAM process, this is not definitely true, some resolution pairs of images may not be accepted for some technical reasons. Empirically, it has been proven that to both satisfy the hardware of the Raspberry Pi, and to ensure the stability of data transmission as well as to ensure the input requirements of RTABMAP, the resolution of color and depth image should be 1280x720, 848x480 in order with 6Hz

frame rate. The configuration of the ROS package launch file with the most basic parameters will be as follows:

```
<include file="$(find realsense2_camera)/launch/includes/nodelet.launch.
xml">
<arg name="depth_width"              value="848"/>
<arg name="depth_height"             value="480"/>
<arg name="enable_depth"             value="true"/>
<arg name="depth_fps"                value="6"/>

<arg name="color_width"              value="1280"/>
<arg name="color_height"             value="720"/>
<arg name="enable_color"             value="true"/>
<arg name="color_fps"                value="6"/>

<arg name="align_depth"              value="true"/>

<arg name="unite_imu_method"         value="linear_interpolation"/>
```

If no configuration is set for the file launch, the default values for the above parameters are as follows:

```
const int IMAGE_WIDTH     = 640;
const int IMAGE_HEIGHT    = 480;
const int IMAGE_FPS       = 30;

const bool ALIGN_DEPTH    = false;
```

In terms of IMU data, the values for the covariance considered during experiments were one, and simultaneously created a unified IMU topic where for each original reading one section is the original and the other, an interpolation.

```
<arg name="unite_imu_method"         value="linear_interpolation"/>
```

The realsense2_camera package, operated by Raspberry, collects data from the camera for the SLAM process and sends data streams to the device responsible for operating the SLAM RTABMAP over a wireless local area network. In addition, the communication between devices in this local network is a two-way communication, through which requests can be made for data transfer request to the Raspberry Pi and receive data in the opposite direction.

4.3.1.2. Data transfer in network

The devices and sensors in the project are designed to be in the same wireless local area network (WLAN) to perform device address management and two-way data transmission between them. Instead of using a traditional local area network with a cable (LAN), using wireless local area network (WLAN) allows data transfer between devices without any physical connection. This allows them to communicate and connect wirelessly within a certain range. When a data transfer process between two devices is established, a message is placed into a packet that includes the message data wrapped between the first signal, the last signal and sent to the other devices on the network infrastructure. NIC will check

the destination address in the first signal of the packet to determine the correct destination, when the signal packet goes to the machine with the destination address, the destination on that machine will copy the signal packet and take the data out of the message packet. Therefore, when it comes to transmitting data between devices on the network, it is important to manage the IP addresses of devices in the system as well as there must be complete, bi-directional connectivity between all pairs of devices, on all ports. Basically, when a device joins the network, an IP address will be assigned to that device, restricting the IP address range can be set by interfering with the configuration of the router. However, this only helps to limit, not to determine a fixed IP address for the device, causing many problems in managing the address of the device.

In the local area network, the router distributes data to different devices. The router is also responsible for assigning IP addresses - more specifically, the DHCP server built-in of router. DHCP is a protocol that automatically assigns a new device an IP address from a group of available IP addresses without any interaction from the user or system administrator.

For Raspberry Pi, either either useful or necessary to be provided with a static IP address. To be able to access the Raspberry Pi on the same address on a private LAN, it must be provided with a static, unique IP address. A static IP address for Raspberry Pi can be used for remote access in SSH network protocols (Secure Shell). This increases the system's flexibility when it is not necessary to connect the required hardware such as a mouse, keyboard or computer monitor to interact with the Raspberry Pi. A static private IP address is also needed because the Raspberry Pi is set up to transmit data from a ROS master, the process can be considered as devices interacting with a server in the network. Therefore the IP address of the device needs to be fixed so that the communication process with ROS master would not be interrupted.
For Raspberry Pi, Raspbian Jessie, or Jessie Lite - current Raspbian operating systems at the moment - provide a DHCP client daemon (DHCPCD) that helps communicate with DHCP servers from routers. The DHCP client daemon's configuration file allows long-term changes to the private IP address of the Raspberry Pi.

To assign an IP address to Raspberry Pi, use the command ‘static ip_address=’ followed by the desired IPv4 address and the suffix ‘/24’ (an abbreviation of the subnet mak 255.255.255.0). In addition, if the Raspberry Pi is connected to the internet via Ethernet or a network cable, use the 'eth0 interface, and wlan interface in case of connecting via Wi-Fi. It also needs to be made clear that the IP address of each device in a network needs to be unique. The Raspberry Pi in this project is connected to the router via Wi-Fi connection and the desired ip address is 192.168.19.11 so DHCP needs to follow the following settings:

```
interface wlan0
```

```
static ip_address=192.168.19.11/24
static routers=192.168.19.1
static domain_name_servers=192.168.19.1
```

Similar to the case of the Raspberry Pi, the image processing computer is also a data transmission device in the network so a static IP setting is also needed. On the same principle as DHCP configuration, the static IP setup implementation on Ubuntu 16.04 running on devices only has a slight difference: in interfaces, the interfaces file is where to add some information about the static IP address on the default DHCP base setting. In the DHCP configuration for the devices in this project, we set the static IP for the image processing device to 192.168.19.90, for the Raspberry Pi collecting data from the realsense D435i camera or 2D LiDAR and Pi performming interactions with UR5 robot arm are 192.168.19.11, 192.169.19.41 and 192.168.19.81 respectively. However, the network address of each device can be changed depending on the structure of the system or the operator's demands.

Once devices on the network have a fixed address, linking ROS on multiple devices via roscore running on ROS master becomes easier and more stable. ROS Master provides naming and registration services with the rest of the nodes in the ROS system. It tracks publishers and subscribers of topics as well as services. The role of the Master is to allow individual ROS nodes to locate each other. Once these nodes have located each other, they communicate with each other on the same level (ROS master).

The master usually runs using the roscore command, roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system. It is essential to have a roscore running in order for ROS nodes to communicate. When working with ROS on multiple devices, it only needs one roscore running on one of those devices, the device responsible for running roscore acts as a master ROS in the networkA note coming from official support documents from ROS, devices should use the same given ROS version because different versions of ROS using the ROS master of one of the versions may cause problems in running few packages due to version/ message type/ dependency incompatibilities. However, different versions of ROS share a common method `export ROS_MASTER_URI` with the IP address of the device running roscore and ROS_IP is the device's own ip address with the bashrc configuration file to establish a link between the device using ROS and roscore running on the ROS master. The ROS master uses port 11311 by default, but it may be changed by passing the `–port` option to roscore and by setting a different port in ROS_MASTER_URI although that ports below 1024 usually require root privileges. Each ROS node sets up a few listening TCP sockets on random ports, one to service xmlrpc calls from the ROS master and other nodes, and one or several for topic connections and services. These ports are randomly allocated by the OS, so it's not possible to know those before chúng được assigned. With a more powerful configuration than the

Raspberry Pi and also the terminal that receives data sent from the camera and sensor, the computer responsible for running RTABMAP to process image data, was be configured to run a common roscore for entire system, become a master ROS for all devices that use ROS in the network.

In general, establishing roscore link on ROS master with other ROS clients will be easier and more stable if the devices in the link have their own static IP. Most modern computer networks, especially small local networks controlled by routers, use DHCP, so setting up static IP for devices on the network is quite simple.

4.3.1.3. RTABMAP-ROS and imu_filter_madgwick

RTABMAP and imu_filter_madgwick are two packages installed on computers responsible for implementing SLAM. Considering remap the magnetometer topic in the imu_filter_madgwick package, when configuring the package to use magnetometer (default), the node waits for matching pairs of messages on topics /imu/data_raw and either /imu/mag or /imu/magnetic_field.

/imu/data:

```
imu_publisher_=nh_.advertise<sensor_msgs::Imu>(ros::names::resolve("imu"
) + "/data", 5);
```

/imu/data_raw:

```
imu_subscriber_.reset(new  ImuSubscriber(nh_,  ros::names::resolve("imu")
+ "/data_raw", queue_size));
```

However, realsense2_camera does not provide magnetic message, so no data was be published on topic / imu / data output of the package. Therefore, the option of using input data for a magnetometer can be eliminated by setting the parameter use_mag to false. In addition, when setting the parameter _publish_tf to true, the package will publish a TF transform that represents the orientation of the IMU, using the frame specified in fixed_frame as the parent frame and the frame given in the input imu message as the child frame. However, because TF is primarily used as a TF base camera published from the realsense2_camera package, the publishing of an additional TF from this package has been omitted to reduce the amount of data and topics.

```
/imu/data_raw := /camera/imu
_use_mag := false
```

On the other hand, in terms of world frames, the fact that IMU devices can measure data in two frames, depending on the design of the manufacturer:

- The sensor frame represents the internal reference frame of the device, which is fixed to the device (ie the coordinate frame moves)
- If orientation estimates are provided (see Data Report), the device world frame represents an external reference to the specified sensor frame. Two of three degrees of freedom of the sensor frame w.r.t., the world frame are usually considered "fixed" through gravity

+ For NED type IMUs, the orientation of the world frame is x-north, y-east, z-down, relative to magnetic north.

+ For ENU type IMUs, the orientation of the world frame is x-east, y-north, z-up, relative to magnetic north.

```
namespace WorldFrame {
            enum WorldFrame { ENU, NED, NWU };
}

if (world_frame == "ned") {
            world_frame_ = WorldFrame::NED;
} else if (world_frame == "nwu"){
            world_frame_ = WorldFrame::NWU;
} else if (world_frame == "enu"){
            world_frame_ = WorldFrame::ENU;
```

ENU is the type of IMUs used in this project `_world_frame := "enu"`. When the world_frame is set to enu (East-North-Up). This means that the X axis of the world frame is pointing East, the Y axis points North, and the Z axis points Up. The orientation field of the IMU message represents the orientation of the IMU frame with respect to this virtual world frame. In the neutral orientation (orientation.xyzw = 0,0,0,1), the IMU frame and the world frame are aligned. This means that in the neutral orientation, the acceleration and magnetometer readings should look like this:

Acceleration: (0, 0, +)
Magnetic field: (0, +, -) (northern hemisphere)
(0, +, +) (southern hemisphere).

In other words, the acceleration vector points towards positive Z (because the IMU frame's Z axis points Up), and the magnetic field vector points towards positive Y (because the IMU frame's Y axis points North).

Finally, navigation for output and input topics, imu_filter_madgwick will automatically subscribe to the topic named / camera/imu connected to the output topic of the IMU data from package realsense2_camera. Similarly, the filtered imu data will be published to topic / rtabmap / imu which matches the topic that rtabmap subscribe to receive IMU data.

```
/imu/data := /rtabmap/imu
```

So basically, imu_filter_madgwick receives data from the IMU sensor built into realsense D435i, after processing, it will automatically forward the output directly to RTABMAP.

With RTABMAP, to get filtered imu data from the imu_filter_magdwick package, it needs to be set up to subscribe to topic / rtabmap / imu with the imu_topic parameter. One note is that when using an external data source for angular rate and the orientation such as IMu for rtabmap, RTABMAP should be told that it will receive external IMU data source, this will set up the package waiting to receive the first message as a signal to execute the SLAM.

Similar to the image data sent from the realsense2_camera package running on the Raspberry Pi via the ROS master connection system. The Raspberry Pi via ROS master sends to the RTABMAP two streams of image data that are registered depth image and RGB camera metadata with a frame rate for both of 6Hz. The depth data stream is sent to the topic /camera/aligned_depth_to_color/image_raw and rgb color image frames are sent to /camera/color/image_raw. In addition, the transfer messages heads of the registered depth image and the RGB camera metadata includes the timestamp so it is possible to set approx_sync parameter to false for rtabmap node. To retrieve image data from these two topics, it is necessary to add the names of these two topics to rtabmap with two parameter depth_topic and rgb_topic:

```
depth_topic:= /camera/aligned_depth_to_color/image_raw
rgb_topic:= /camera/color/image_raw
```

The topology of the nodes in rtabmap can be seen in the figure 32. The data requested from the first rtabmap can be referred to as TF to determine the position of the sensors related to the system's base. Odometry is also an important data in the SLAM process of rtabmap, as mentioned, if the configuration uses only rgbd camera, odometry will be odometry generated from rtabmap. However, odometry sources from any other external source can be used, either 3DoF or 6DoF.

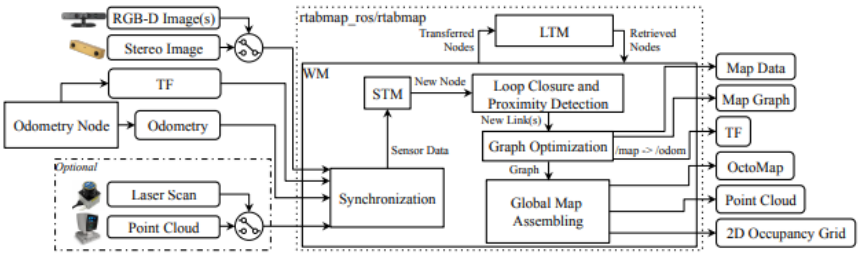


Figure 32. Block diagram of rtabmap ROS node

Optional inputs are laser scanning from a 2D overlay or a point cloud from a 3D overlay. All messages from these inputs are then synchronized to the timestamp and delivered to the SLAM graphing algorithm. With optimized charts, OctoMap, Point Cloud and 2D Occupancy Grid outputs can be assembled and published to external modules. Geometrical correction for localized robots in the map frame is also available via tf / map → / odom. (Foote, 2013)

In general, this system structure can be applied appropriately and steadily to handheld settings. The quality of mesh created from point clouds shows that the image quality, good color and size of the concrete slab were reconstructed in the 3D model relatively accurately compared to the actual size of the object. Comparison results can be monitored in Experimental Results section. In addition, if the scanned area of the concrete slab is not too large and is within the Field of view area of the Realsense camera, it is possible to use the same hardware system and handheld settings to use as a Fixed-mounted scanning system.

However, on the other hand, we have studied various experiments and thereby come up with solutions that help to improve some of the drawbacks of this setting. Because the device will be moved manually, the movement of the device is continuous. Even if the intention of the user is to stand still, the unstable vibration of the human arm still leads to the image obtained as well as the position of the device that is constantly changing. This constantly changing image data stream should not be interrupted for too long which can cause loss image or localization of the system in space due to the lack of image features for RTABMAP. n other words, technically, If the camera is moved too fast, there will be less features to match between successive frames. RGB images can be also blurry depending of the camera, which graduated the number of good visual matches between frames. Moving the camera at a moderate speed but the latency or loss of data caused by the data transmission, in this case is from the Raspberry Pi to the image processing device can also cause the same situation as the camera move at too high speed. In addition, the burden of data transmission also causes misalignment between the depth and RGB images frame, if the depth of the visual features is wrong, this would have a negative impact on the odometry.

In terms of the burden of data transmission, the construction approach of designing a handheld 3D scanning system may face some data transmission speed limitations in theory when data collection and image processing are involved on two different devices. Generally with a download speed of about 37 Mbps and an upload speed of about 25 Mbps of the local network used in the project, it is approviate to perform hand scanning with average moving speed. Though handheld scanning was doable in the case of this project prototype, we developed some methods to improve the overload problem in data transmission. The first is an established method where instead of raw rgb and depth image data is sent from the Raspberry Pi to an image processing device separately as in the general system structure, they were preprocessed so that they are combined into a single message, rgb-d image data. This solution helped reducing image data loss as well as timestamp deviation of frames during data transmission. The second solution is recording data from collecting device (Raspberry Pi) and sending record file to image processing device, by this method, we can completely avoid losing data on the line.

4.3.2 Synchronizing RGB-D image message

The image data for the SLAM process provided by the realsense2_camera package splits into two separate streams for color and depth images. These two image streams are marked with the timestamp at the time of being collected, RTABMAP uses this timestamp to identify pairs of color images and the corresponding depth at a time. However, in the process of sending data over the wireless network in the network, it is possible that these data pairs do not arrive at the final point at the same time but will be deviated a certain amount of time. This makes the SLAM process of RTABMAP slower because it will wait for the data with the same timestamp as the received data to continue processing. And being slower too long sometimes leads to the camera moving to a different perspective in the environment but the data processing is outdated, resulting in the loss of features in the image. In addition, in the process of sending data, it is possible that the data will be lost, causing the frame to be dropped, affecting the scanning process. Therefore, one of the solutions to resolve this problem is to pair the image and depth data into a single rgbd data, and this data will be transmitted to topics through the ROS master system by compressed messages. This solution solves not only the problem of data arriving the different time or losing data on the transmission line because now two separate data becomes one, but also reduces the data transfer traffic when data is now transmitted in compressed form.

To implement this solution, the rtabmap package also requires a successful installation on the Raspberry Pi, although this installation is quite difficult. In rtabmap, node rtabmap_ros / rgbd_sync implements synchronize RGB, depth and camera_info messages into a single message. It is possible to use subscribe_rgbd to make rtabmap or odometry nodes subscribing to this message instead. This model requires 3 required inputs: RGB image stream, registered depth image stream and RGB camera metadata. All three image data are package pubs realsense2_camera into topics corresponding to its settings. In the setup of this project, the configuration for the most important basic parameters for nodelet rgbd_sync would be as follows:

Điền cho phù hợp

```
<remap     if="$(arg compressed)" from="rgbd_image" to="$(arg rgbd_topic
)/compressed"/>
<remap     if="$(arg compressed)" from="$(arg rgbd_topic)/compressed_rel
ay" to="$(arg rgbd_topic_relay)"/>
<remap unless="$(arg compressed)" from="rgbd_image" to="$(arg rgbd_topic
)"/>
<param if="$(arg compressed)" name="uncompress" value="true"/>
```

By setting this nodelet rgbd_sync to pre-process two separate data streams of rgb and depth before sending them to other devices on the

same network, the burden on the previous data link can be significantly reduced

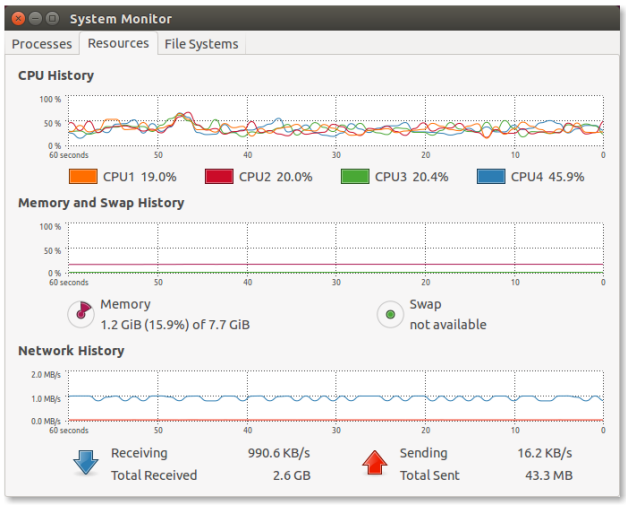


Figure 33. The System Monitor of Ubuntu shows that the bandwidth used was approximately 1 MB / s when using the rgbd synchronization technique

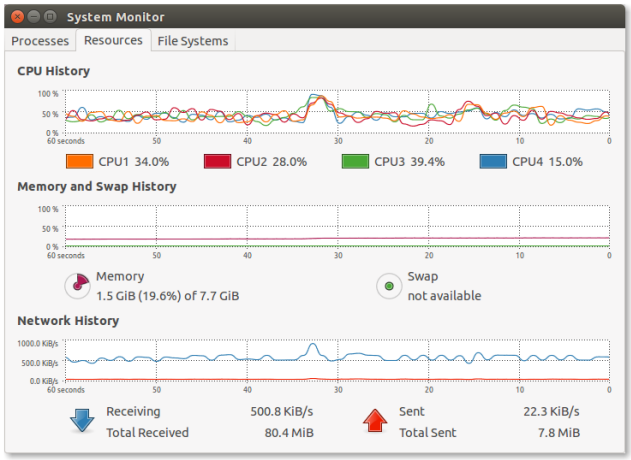


Figure 34. System Monitor of Ubuntu shows that the bandwidth used was about 500 KiB / s with the help of rgbd sync nodelet

4.3.3 Recording data for later mapping

Although the preprocessing data and sending in compressed form solution minimizes the problems that can be encountered when performing remote data processing. However, there is some possibility that the manufacturer does not have a network available for the system or for some reason the bandwidth of the network connection is very small. In this case, it is quite complicate to perform remote mapping directly, the solution to this problem is to record all data from a running ROS system into a .bag file., and then to play back the data to produce similar behavior in a running system. A bag file is considered as a storing bag of messages transferred between topics, which is recorded during the operating session of ROS packages. Data is stored in the bag file under binary format. The design of the rosbag allows the storage of published messages on topics at high speed, minimizing data loss during recording, an essensial aspect. n addition, the rosbag package allows

compressing the file on the fly with the bz2 algorithm that can be implemented by the flag -j. A special thing is that this package allows to store messages from the desired topics, not necessarily all. The same thing is done with the rosbag file replay process when it can be set up so that replays are executed only on the desired topics (Anil Mahtani). The messages belonging to the topics that need to be recored in this setting are also the messages that are required to perform SLAM in RTABMAP. These topics can be listed as /camera/aligned_depth_to_color/image_raw of registered depth image, /camera/color/image_raw of RGB camera metadata, /camera/color/camera_info and /rtabmap/imu of IMU Bosch BMI055. The recording of all messages from the above topics can be done by command:
rosbag record -O my_bagfile_1.bag /camera/aligned_depth_to_color/camera_info camera/aligned_depth_to_color/image_raw /camera/color/camera_info /camera/color/image_raw /camera/imu /camera/imu_info /tf_static". This file can then be sent to another device to perform SLAM on RTABMAP at any time. This is convenient and avoids the burden of data transmission caused when performing remote mapping close to real time.

4.4 **Hardware installation**

The main devices used in handheld setups are the realsense D435i camera, Raspberry Pi which collects image data from the camera via USB 3.0 and forwards data to the SLAM-processing computer in the network via a Local area network is distributed by a WLAN router. Because using the Raspberry Pi to receive and transfer image data via WLAN allows the hardware design of the Handheld system to be independent and separate from low-mobility computer hardware. This part of the article will focus on explaining the basic design of handhelds including the highly customizable camera and Raspberry Pi with the ability to integrate into other types of configuration designs such as the UR5 end-effector, or used as a stationary scanner.

4.4.1 Raspberry Pi

The Raspberry Pi is basically a complete hardware structure with necessary data transfer ports such as USB 3.0, Ethernet as well as USB-C power supply. However, during the research, some new hardware was added to solve encountered issues, such as protecting the Raspberry Pi from external impacts or solving the overheating problem during operating, especially happened with Raspberry Pi 4.

With Raspberry Pi 4 installed ROS kinetic on the latest raspbian Buster platform, the actual operation shows that the operating temperature of Pi 4 was quite high, about 10 degrees higher than the Raspberry Pi version 3 when performing the same task. In idle mode, the Rasppberry Pi does not perform a task at all (CPU is almost 0%), does not connect to the screen (using only SSH) but the temperature has reached about 70 degrees Celsius (measured by temperature monitor or use the command line) and always remain around 70 ° C temperature. When running the realsense package to collect data from the camera, the temperature quickly increased to more than 80 degrees. Any electronic device operating at such a high temperature may cause a decrease in the power of the components, thereby reducing the effectiveness of the device, in addition to reducing the device's durability. when used for a long time. Therefore, solutions to lower Pi's normal operating temperature have been considered and applied.

During operation, the CPU is placed in the SoC (system on chip) which generates most of the heat of the Raspberry Pi single-board computer. However, it is also possible to implement thermal support for the memory chip and ethernet / USB controller, although it does not increase much performance, as they can only reduce the overall temperature to about one degree. Therefore, a thermal support kit for Pi CPU has been evaluated as sufficient to reduce the overall temperature of the device. The heatsink applied to the CPU takes heat out of the CPU, into the heat sink, allowing heat to enter the air, cooling the CPU. A radiator with aluminum heat sinks has been mounted on the SoC. Although most of the heat sinks available for the Raspberry Pi have thermal tape affixed to the bottom, a layer of thermal paste has been used instead because thermal paste has better conductivity.

In addition, an accessory has been added to increase the heat dissipation ability of the SoC is the fan. A 30x30 (mm) blower is placed on the CPU part of the Raspberry Pi to ensure maximum airflow is provided. The fan is mounted via nuts and bolts with a plastic frame surrounding the Pi, which makes it easier to mount the fan and also protects the device from external physical damage. The fans draw power directly from the Pi's General Purpose Input / Output pins, so there's no need for any external power supply.

4.4.2 Handheld frame

One of the project's overall goals is to design a system with high mobility that is used in many different settings with different scanning targets and environments such as being able to integrate LiDAR into the system or easily integrated into the UR5 to act as an end-effector etc. A handheld framework, based on this criterion, has been designed with 3D support mechanical design tools of Inventor CAD software. The material used to

print 3D frames is PETG filament. PET is the most commonly used plastic material in the world, PETG is PET added with a glycol component (the letter G). This chemical component helps prevent high temperature crystallization with PET. Thanks to this, the product has a higher thermal endurance. In addition, mechanical properties also proved to be superior to PET and far more than traditional PLA ABS 3d printing filaments. Thanks to that, the real structure of 3D printed frame is stable.

In terms of devices used in handhelds in particular, the chassis should be designed to be able to integrate Realsense D435i and Raspberry Pi cameras. A suitable mounting position for the camera must be taken into account seriously. Approviate camera placement can prevent the camera from having a wide viewing angle that retracts into the corners of the handheld frame. So the camera placement is designed so that when the camera is attached, its lens is at the protruding position to the front. In addition, the camera position also needs to be stable and firm to help the camera capture the most accurate frames, avoiding camera shake that makes scanning difficult and performing SLAM, also to have an ability to protect the camera from physical impacts from the environment such as bumping or falling because the trajectory moves in the space of the handheld device is due to the grip and movement of the human hand. Two small bore holes in place coincide with the two screw holes available on the camera that allow the use of screws to fix the frame and the camera into a fixed unit. In addition, during use, the camera emits a significant amount of heat to the outside, making the lack of ventilation in places where the airflow of the device drains can increase the overall temperature of the camera. The solution is to have two grooves cut in the upper and lower wall so that the camera's heat slots have space to direct heat to the air. Finally, the calculation for the ability to dismantle the camera, the two left and right camera fixing walls along the x axis has been omitted so that the camera can be removed or inserted by sliding into the frame from either side. The placement and the fixed frame camera were designed on Inventor as figure 35.

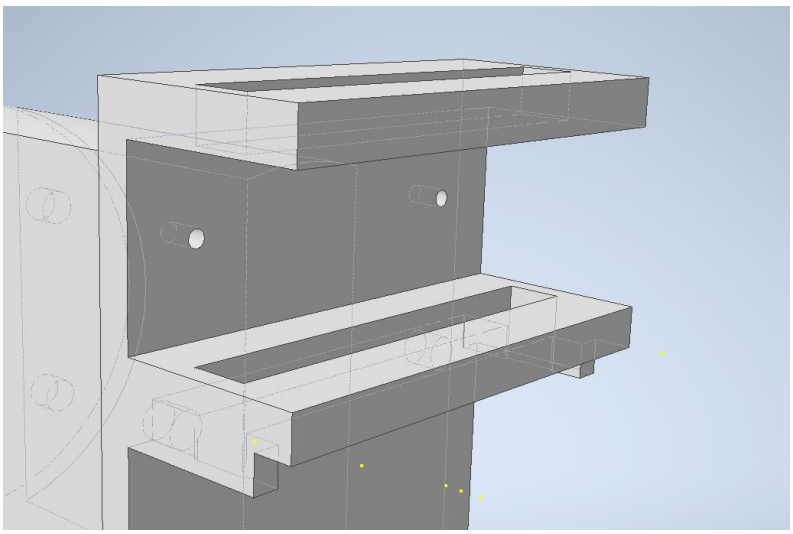


Figure 35. Fixed camera location on handheld frame

The Raspberry Pi has a hard, box-shaped case that can easily be attached to a flat surface highlighted by a green background as shown in figure 36.

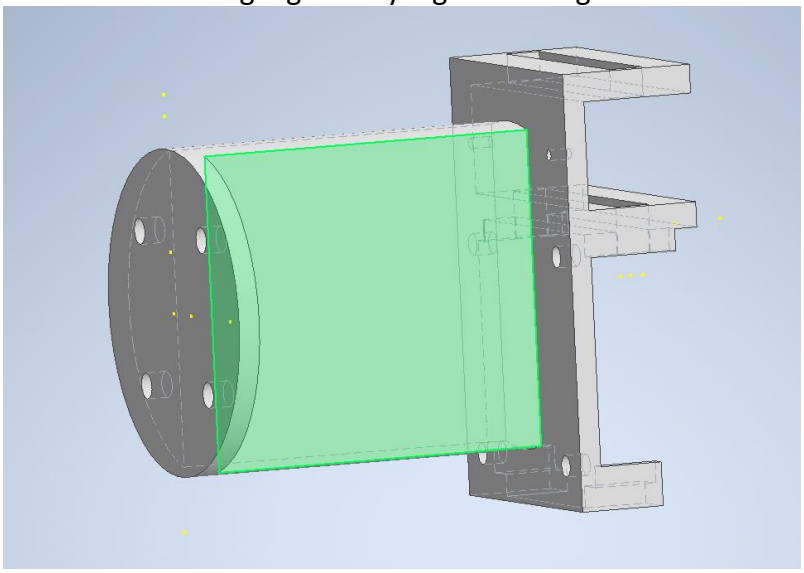


Figure 36. The plane designed to attach the Raspberry Pi

This plane is equally designed on both sides of the frame so that the choice of placement of the Raspberry Pi can be more proactive depending on the USB cable connection between the Pi and the camera.

Finally, considering the scalability of the system, it is about the ability to integrate more devices or the ability to integrate into another device. This project further researches object scanning configurations that use LiDAR as an external source of odometry for SLAM processes. The plane directly below the camera's fixed frame is the LiDAR reserved plane when needed. Four 3 mm-diameter screw holes in four flat corners are designed to be position-compatible with 4 screw holes on the LiDAR anchor frame to make mounting and disassembling the two parts easier as shown in figure 37.

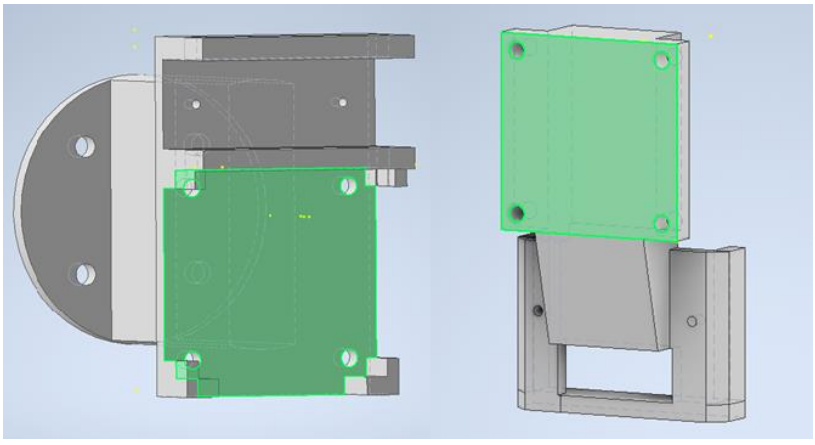


Figure 37. The designed LiDAR frame is able to link or detach easily with handheld frames

The LiDAR camera will be attached to the frame via two 1mm diameter screw holes, the power cord and Ethernet cord from LiDAR will be passed

through a rectangular shaped groove placed just on the contact plane between it and the frame.

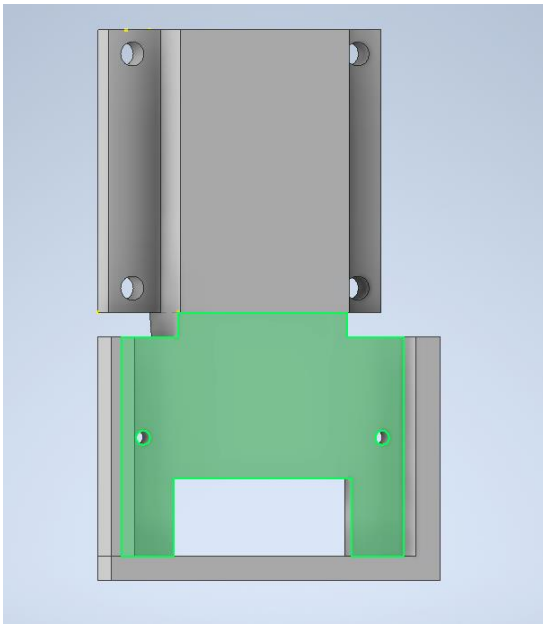


Figure 38. LiDAR sensor mounting frame

On the other hand, the handheld design also allows the frame to be attached directly to the final twist joint of the UR5 machine arm by a circular bottom plane with a diameter equal to the diameter of the end-effector's connecting plane on the UR5. Like other equipment parts integrated into the frame, the mounting of the frame into the machine arm is also done through the structure of the screw. With 4 screws with a large diameter of 6 mm, mounting the frame on the arm of the machine becomes sturdy and reliable.

Figure 39. Connecting plane of UR5 end-effector

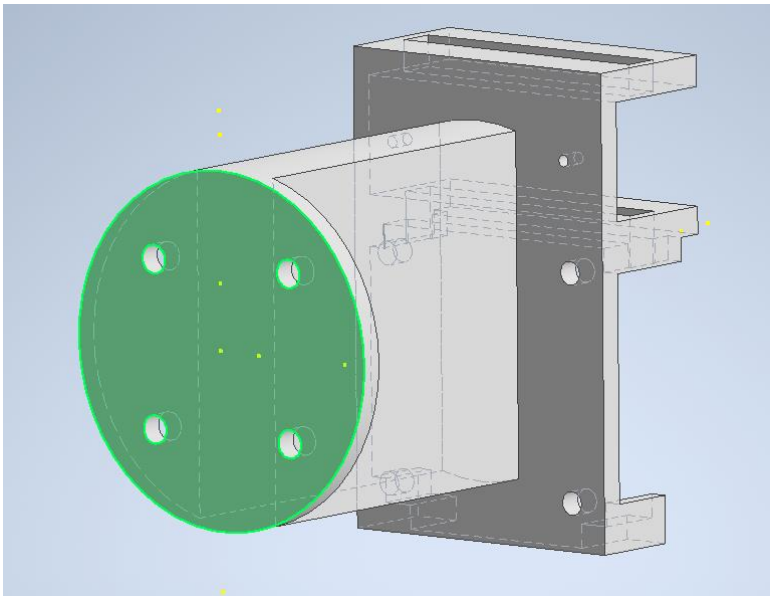


Figure 40. Circular plane with a diameter equal to the diameter of the end-effectors connecting plane on the UR5 allow handheld frame integrating easily to robot arm

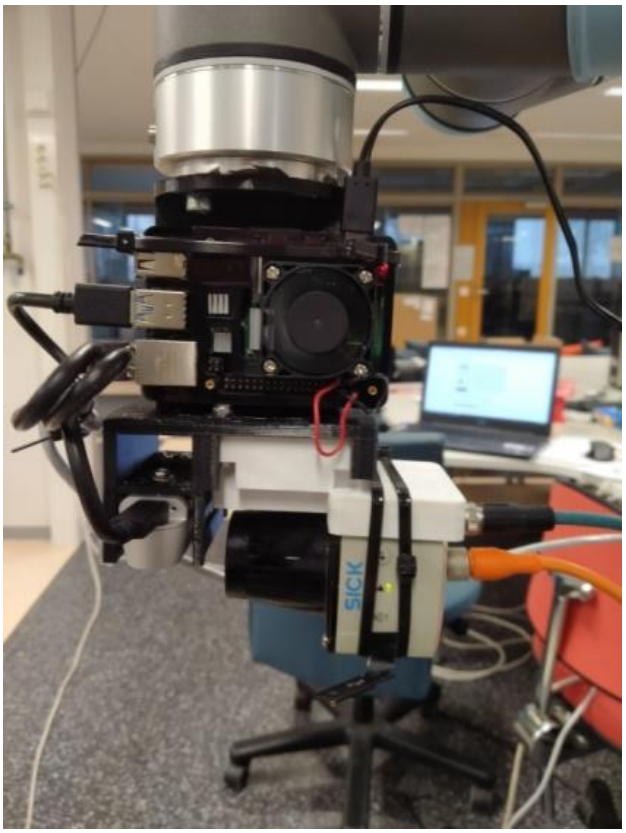


Figure 41. 3D handheld scanner mouted on the flange plate of Robot arm with RGB-D camera located on the vertical axis with 2D LiDAR sensor. Raspberry Pi in the black case was used to collect and forward image and laser data from camera and sensor to remote PC.

4.5 Horizontal planar scanning with external odometry

The handheld design can be used in another scenario: horizontal planar scanning. The scenario of these approaches assumes a planar lidar

mounted on the scanner moving on a 2D plane which limits the estimated robot poses to 3 DoF (2D position and an orientation angle). RGB images provide dense texture but lack depth information which LiDAR performs better. Therefore, LiDAR could be ultilized to measure the distance of the system to the external environment, thereby performing positioning in 3D environment instead of RGB-D Camera. (Chen Fu, 2019)

Visual odometry in rtabmap works in the following order: to calculate odometry, the algorithm uses visual indications derived from an RGB image and depth data from a depth map. Using the matching of visual signs (matching) between two images, the RANSAC algorithm calculates the transformation between successive frames. It is able to configure the internal parameters of the algorithm, such as the maximum distance between the inlayers, the maximum number of visual signs for matching, the number of iterations in the RANSAC / ICP method. In general, experiments with visual rtabmap odometry showed that the algorithm works quickly, without delays and accurately determines the position of the camera relative to the scene. The robot pose estimated by RTAB-Map has 6 DoF, which makes it suitable for non-planar (e.g. drones) robots. However, as mentioned, when the scanning system moves only on the 2D plane, the pose of the system is limited to 3 DoF. This is suitable for using LiDAR 2D, providing more accurate distance measurement of the object using time of fligh than rgb-d camera (Chen Fu, 2019) to create an external low-drift odometry source for Rtabmap instead of visual odometry. To collect data from the LiDAR sensor LM561 and use this data source to create odometry data, the two ros packages used in turn are sick_scan and laser_scan_matcher.

4.5.1 Sick_scan

As introduced in the System Overview section, sick_scan is an open source ros package, installed on the Raspberry Pi to collect laser scan data stream from the LiDAR sensor. In this project, sick LM561 is 2D LiDAR used, this sensor allows data exchange via Ethernet cable and thus, it allows users to configure a static IP address. Setting the IP address can be done via SOPAS ET software or the sick_scan package can be used. However, the Ros package IP change tool requires that the device's IP address be pre-determined to identify the correct device on the network to make changes. Therefore, to set up a brand new scanner, it is recommended to use the two tools "sick_generic_device_finder.py" to find the scanner in the network and the launch file sick_new_ip.launch to set a new IP address. However, if further settings are to be saved that cannot be made via ROS parameters, it is recommend to use the Windows tool "Sopas ET" from SICK.

One note when using the sick_generic_device_finder.py tool to find the IP address of a device is if it has more than one network interface it may

happen that no scanner is found because the broadcast ip address does not match and the discovery packets are sent to the wrong interface. To fix this problem change the parameter `UDP_IP = "192.168.0.255"` to the broadcast address that ifconfig returns for your network interface e.g. "192.168.178.255".

Set the launch file to change the IP for sick LiDAR as follows.

```
<launch>
<arg name="hostname" default="192.168.0.1" />
<arg name="new_IP" default="192.168.0.2" />
<node   name="sick_new_ip"   pkg="sick_scan"   type="sick_generic_caller"
respawn="false" output="screen">
<param name="scanner_type" type="string" value="sick_tim_5xx" />
<param name="use_binary_protocol" type="bool" value="True" />
<param name="hostname" type="string" value="$(arg hostname)" />
<param name="port" type="string" value="2112" />
<param name="timelimit" type="int" value="5" />
<param name="new_IP_address" type="string" value="$(arg new_IP)"/>
</node>
</launch>
```

The IP address of LiDAR sensor in this project was set to be 192.168.0.55
Once the device has the device's IP address, using the sick_scan package to collect laser scan data from the LiDAR sensor is quite simple with the command structure:

```
roslaunch <launch-file> hostname:=<ip-address>
```

In particular, the launch file used for the LM5xx series is sick_tim_5xx.launch. PointCloud and Laser Scan data streams will be published on two topics named /cloud và /scan

4.5.2 Laser_scan_matcher

Laser_scan_matcher is the ROS package used in the project to create the source of odometry from LiDAR depth data. The laser_scan_matcher package is an incremental laser scan registration tool. The package allows to scan match between consecutive sensor_msgs/LaserScan messages, and publish the estimated position of the laser as a geometry_msgs/Pose2D or a tf transform. The package can be used without any odometry estimation provided by other sensors. Thus, it can serve as a stand-alone odometry estimator. Alternatively, you can provide several types of odometry input to improve the registration speed and accuracy. Moreover, IMU input if provided for the package, allowing significantly improve convergence speed for rotational motion. The addition of an IMU input is thus highly recommended. Ngoài ra, để giảm bớt a slow drift of the pose of the robot due to inevitable noise, laser_scan_matcher implement keyframe-based matching. The change in pose is calculated between the current laser scan and a "keyframe" scan. The keyframe scan is updated after the robot moves a certain distance. Thus, if the robot is standing still, the keyframe scan will not change, and the pose will remain more drift free. Setting the tolerance for updating

the keyframe can be achieved via the kf_dist_linear and kf_dist_angular parameters. Their default values give a more robust performance, both while standing still and moving.

However, one limitation of laser_scan_matcher is that scan matcher is not measuring velocity directly. Although scan matcher is not measuring velocity directly, it is computing the delta in the laser position, which we could combine with the time between laser scans to produce a rough estimate of velocity, subject to jitter in laser frequency and any errors in the scan matching. Since laser_scan_matcher does not publish stamped poses, we have to modify laser_scan_matcher directly to compute this output. LaserScan has a timestamp in the message header that we can use to compute the time between scans.

```
laserScanToLDP(scan_msg, prev_ldp_scan_);
last_icp_time_ = scan_msg->header.stamp;
```

Adding timestamp into output messages header:

```
ros::Time new_icp_time = ros::Time::now();
ros::Duration dur = new_icp_time - last_icp_time_;

if (publish_tf_)
                {
                tf::StampedTransform transform_msg (w2b_, time, fixed_frame
                _, base_frame_);
                tf_broadcaster_.sendTransform (transform_msg);
                }
if (publish_vel_)
                {
                odom.header.stamp = time;
                …
```

Basic settings the package are as follows:

```
                <include file="$(find realsense2_camera)/launch/includes/no
delet.launch.xml">
<arg name="depth_width"              value="848"/>
<arg name="depth_height"             value="480"/>

<node             pkg="tf"             type="static_transform_publisher"
name="base_link_to_laser" args="0.0 0.0 0.0 0.0 0.0 0.0 /base_link /laser
40" />
 <node pkg="lidar" type="laser_scan_publisher" name="laser_scan_publisher"
args=""/>
<group if="$(arg publish_covariance)">
                <param  name="laser_scan_matcher_node/do_compute_covariance"
                value="1"/>
                <param
                name="laser_scan_matcher_node/publish_pose_with_covariance"
                value="true"/>
                <param
                name="laser_scan_matcher_node/publish_pose_with_covariance_
                stamped" value="true"/>
 </group>
 <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
                name="laser_scan_matcher_node" output="screen">
                <param name="max_iterations" value="10"/>
 </node>
```

The output odometry data source will be sent to the topic named / odom, this topic name is completely customizable by remaping to a topic with the desired name. This odometry data will be forwarded to RTABMAP as an independent external source of geometry for SLAM process.

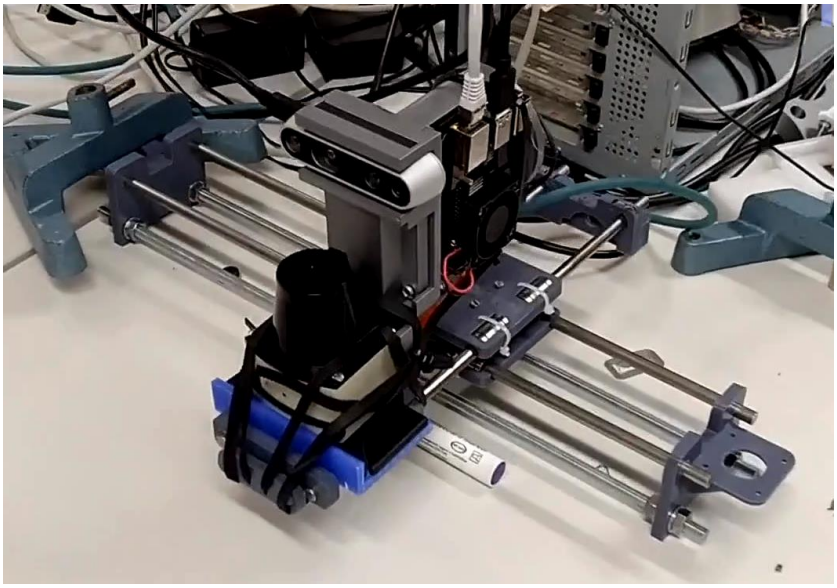


Figure 42. Planar scanning with a slider support

4.5.3 Robotic scan

Cobot configuration is considered as a hands-free solution for scanning featuring the 3D scanner mounted to a UR5 robotic arm. Unlike the handheld 3D scanning devices that have to be operated manually, this 3D scanning arm not only captures data automatically, but also does it faster, more accurately than a person. The most important thing when using the UR5 robot in this project is the ability to reduce or totally eliminate the need for time-consuming manual labor thanks to trajectory and control planning for robot manipulations.

There are many ways to set up a UR5 arm for use with ROS. In this project, the ros package ur_modern_driver is used to provide a python interface that allows pre-programming trajectory for UR5 through scripts used in parallel with the API provided by the package. Besides that, Moveit to build up a representation of their environment using data fused from 3D and other sensors, generate motion plans that effectively and safely move the robot around in the environment, and execute the motion plans while constantly monitoring the environment for changes.

One note is that when running a connection between ROS on Ubuntu or a Raspberry raspbian with UR5, it is required to open reverse port 50001 for communication:

```
sudo ufw disable
sudo ufw enable
sudo ufw allow 50001/tcp
```

When you want to connect to two different robots in the same network, it is possible to open another port, for example, port 50002.

```
sudo ufw allow 50002/tcp
```

4.5.3.1. Ur_modern_driver

Because the ros_control-based approach has been used, it is necessary to install the additional ur_driver package in the same workspace as ur_modern_driver. Ur_driver acts as an add-on to the ur_modern_driver package by providing this package with two smaller packages, ur5_moveit_config and ur_description.

Ur5_moveit_config contains Moveit configuration file of Universal Robot manipulators. To use ros_control together with MoveIt, it requires add the desired controller to the controllers.yaml in the ur5_moveit_config/config folder. Controller configuration list can be added to the end of the .yaml file with the following structure:

```
controller_list:
-name: /vel_based_pos_traj_controller #or /pos_based_pos_traj_controller
action_ns: follow_joint_trajectory
type: FollowJointTrajectory
default: true
joints:
-shoulder_pan_joint
-shoulder_lift_joint
-elbow_joint
-wrist_1_joint
-wrist_2_joint
-wrist_3_joint
```

In addition, the driver currently supports two position trajectory controllers. Tasks performed by the velocity interface are performed with higher speed, lower latency, and simpler path planning. The orbital interface is the action interface, which takes the target and verifies that the trajectory is valid and transmits an intermittent matching position to a custom script that runs on the controller. Only one of them can be running at the same time. When setting up the ur_modern_driver package, switching between these two interfaces depending on different usage can be done by running a rosservice as follows:

```
rosservice call /universal_robot/controller_manager/switch_controller "s
tart_controllers:
- 'vel_based_pos_traj_controller'
stop_controllers:
- 'pos_based_pos_traj_controller'
strictness: 1"
```

After the installation is complete, the package is ready to connect to the UR5. Call an instance of the ur5_ros_control.launch package with the manipulator's ip address in the robot_ip: = parameter so the package would link to the robot with the corresponding IP. Note that both robot and Raspberry Pi must be in the same network. Trajectory programming for manipulator can be done via python interface of package.

```
Def move_repeated():
  g = FollowJointTrajectoryGoal()
  g.trajectory = JointTrajectory()
  g.trajectory.joint_names = JOINT_NAMES
    try:
    joint_states = rospy.wait_for_message("joint_states", JointState)
    joints_pos = joint_states.position
    d = 2.0
    g.trajectory.points = [JointTrajectoryPoint(positions=joints_pos,
    velocities=[0]*6, time_from_start=rospy.Duration(0.0))]
    for i in range(10):
      g.trajectory.points.append(
        JointTrajectoryPoint(positions=Q1, velocities=[0]*6,
        time_from_start=rospy.Duration(d)))
      d += 1
      g.trajectory.points.append(
        JointTrajectoryPoint(positions=Q2, velocities=[0]*6,
        time_from_start=rospy.Duration(d)))
      d += 1
      g.trajectory.points.append(
        JointTrajectoryPoint(positions=Q3, velocities=[0]*6,
        time_from_start=rospy.Duration(d)))
      d += 2
    client.send_goal(g)
    client.wait_for_result()
except KeyboardInterrupt:
    client.cancel_goal()
    raise
except:
    raise
```

In particular, `JointTrajectoryPoint` was used to declare parameters for message trajectory_msgs/JointTrajectoryPoint Message with the values positions, velocities, accelerations, time_from_start respectively. Each trajectory point specifies either positions[velocities[, accelerations]] or positions for the trajectory to be executed. All specified values are in the same order as the joint names in JointTrajectory.msg. For clarification, time_from_start is relative to trajectory.header.stamp. Each trajectory point's time_from_start must be greater than the last. In the Python code above, the variable d was the variable used for the value of time_from_start. The velocities specify the joint velocities the joints have at that trajectory point and the velocities should be all 0 for the first trajectory point. Moreover, trajectories with unspecified accelerations use cubic spline interpolation, hence have no acceleration continuity. Trajectories specifying accelerations use quintic splines and guarantee acceleration continuity. Therefore, the accelerations are regularly all 0 in practice. When running this python script, the robot will move to desired pose in turn. All pose of robot will be published on topics.

4.5.3.2. Moveit

Moveit was used in this project to do path planning for ur5, MoveIt! comes with a plugin for the ROS Visualizer (RViz). The plugin allows to setup an environment in which the robot will work, generate plans, visualize the output and interact directly with a visualized robot
Moveit is partially integrated in the ur_modern_driver package. When performing the pre-setup steps as well as running the ur_modern_driver package as above, it was part of the Moveit feature path planning feature.

Once connected to the UR5 robot, start the Moveit node ur5_moveit_planning_execution for motion planning with sim: = true to start Moveit as a simulation. This node, when called, would remap the action controller node with the name of the controller declared in controllers.yaml, which means that MoveIt is trying to find the arm_controller/follow_joint_trajectory action client specified by controller.yaml. Then to make the moveit connection with rviz, set up and run a launch file with the following configuration:

```
<launch>
  <arg name="debug" default="false" />
  <arg unless="$(arg debug)" name="launch_prefix" value="" />
  <arg     if="$(arg debug)" name="launch_prefix" value="gdb --ex run --
args" />
  <arg name="config" default="false" />
  <arg unless="$(arg config)" name="command_args" value="" />
  <arg         if="$(arg  config)"  name="command_args"  value="-d  $(find
ur5_moveit_config)/launch/moveit.rviz" />
  <node    name="$(anon    rviz)"    launch-prefix="$(arg    launch_prefix)"
pkg="rviz" type="rviz" respawn="false"
  args="$(arg command_args)" output="screen">
  <rosparam                   command="load"                  file="$(find
ur5_moveit_config)/config/kinematics.yaml"/>
  </node>
</launch>
```

To use the Motion Planning Plugin at the very first time, it is necessary to add Rviz Displays Tab. From the moveit_ros_visualization folder, choose "MotionPlanning" as the DisplayType.

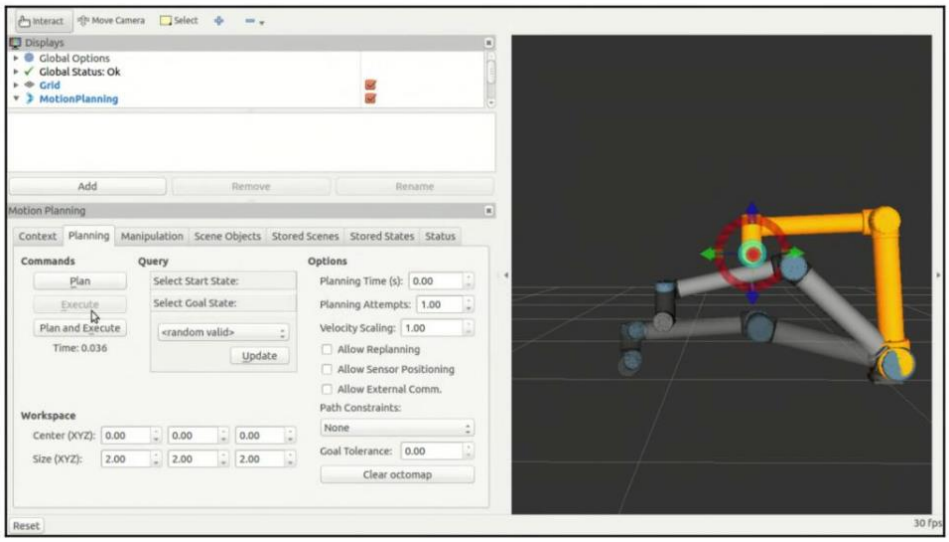


Figure 43. Path planning interface of ur_modern_driver running on RVIZ

Press Interact in the top menu of rviz, a couple of interactive markers appear for the UR5 robot. One marker (orange colored) can be used to set the “Goal State” for motion planning. Another marker corresponding to a green colored can be used to set the “Start State” for motion planning. In Rviz's simulation environment, user can move the end-effector robot to the desired location using the Plan button. When planning the next position of the end-effector as well as the position of the joints, use the "Execute" button or the "Plan and Execute" button to send the tranjectory to the robot. Robot arm in Rviz simulation environment as well as real manipulator should perform the same motion.

5 EXPERIMENTAL RESULTS

The prototype developed in this project has successfully used the rgb-d camera as a central device to perform reconstruction of precasted concrete slab surface. The polygin file format (.ply) output of the scanning system contains mesh data of hollow slabs, were be used to perform geometric measurements where the results of these measurements used to conclude whether the product has qualified enough to ship or not.

Polygon file format is a format designed to store three-dimensional data from 3D scanners. There are softwares currently on the market that support to render this file. Among them is the open source platform CloudCompare, which is used to make simple measurements on an 3D model of the concrete slab for the purpose of estimating the accuracy of the output of the scanning system in this project rather than using as an official tool for performing product quality checks. For manual measurements on actual products, two actual distances from four steel

strands were used as a reference for comparative measurements on the 3D model that can be seen in the figure 44 below.

Figure 44. The distance between two successive steel reinforcement on two hallow slab (A) and (B) products was measured with lengths of 26.5cm and 23.7cm, respectively.

First, in terms of the results of the Handheld setup which has the hardware design and software setup was the basis structure of other extensive settings. Handheld scanning method was considered as the most complicated approach in 3D scanning because of the instability of human arm movements when performing scan. Therefore, two 3D models achieved from the scanning process using the handheld setup were used to perform measurement procedure in order to check the accuracy compared to the reference length.

After installing CloudCompare, to start the software in Ubuntu, enter the command "cloudcompare.CloudCompare" into the terminal. The 3D model of the two cross-sectional concrete slab scanned by the handheld setting has been successfully exported to CloudCompare, as can be seen in the figure 45.

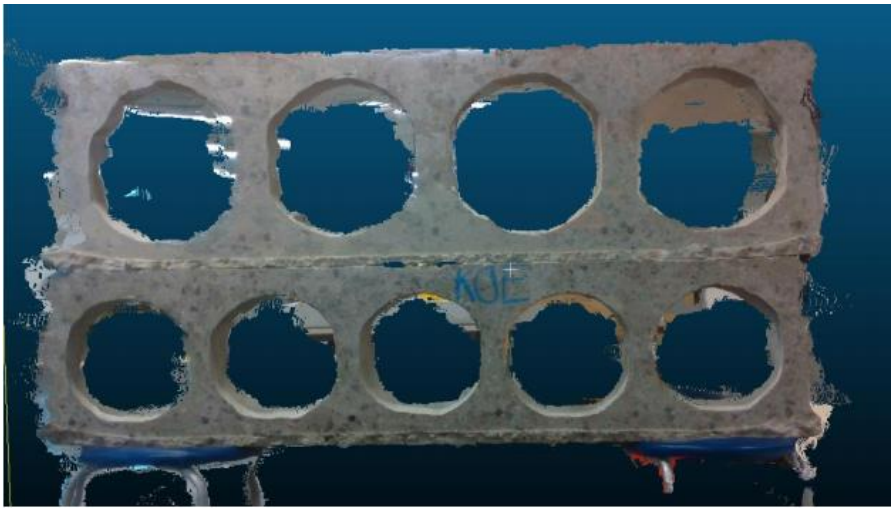


Figure 45. 3D model of cross section of two hollow slabs concrete products scanned using handheld setup loaded in CloudCompare

PointCloud support software provides many useful tools for processing and interacting with 3D data files. In particular, Point picking is a tool to help perform geometric measurements on 3D models. The main purpose of this tool is to allow users to select one, two or three points to get a variety of information (most notably the distance between the two points). This tool is accessible via the main upper toolbar or the 'Tools > Point picking' menu. With the icon , user can pick two points and make a 'distance' label appear. The distance between two reinforcing steels on the product (A) and (B) as shown in figure 46 on the handheld scan model was 26.73cm and 23.78cm respectively, as shown in the picture below.

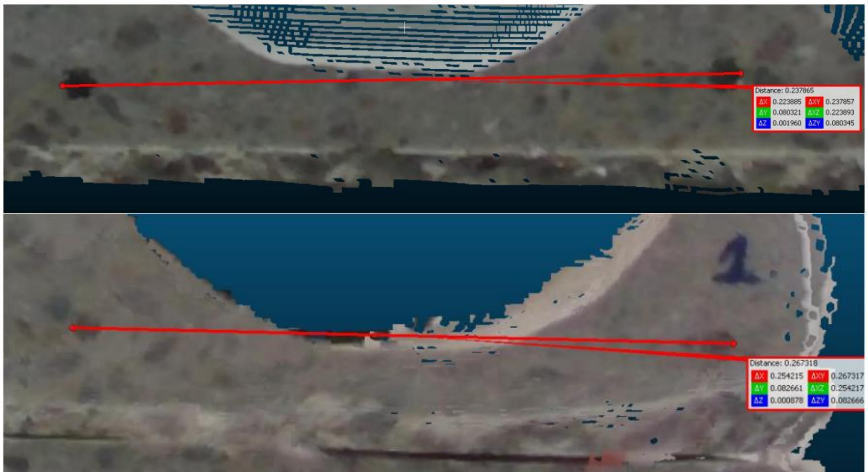


Figure 46. The distance between the two steel strands on the hollow slabs model of the hollow s lab A and B were scanned by the handheld device

Expanded on the basis of handheld design structure, the scanning system supported by the UR5 robot also yields almost equivalent results to the distance between the two steel strands on the hollow slab type (A) on the model measured 26.38cm and the distance measured on the model of type (B) is 23.51cm

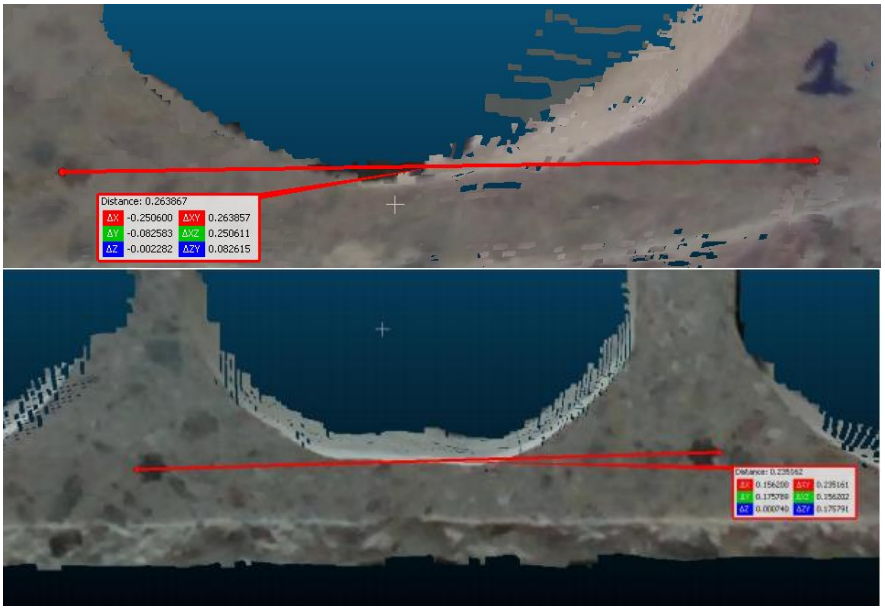


Figure 47. The length between two reinforcing steels was measured on the 3D model of hollow slab type A and B with the scanning system supported by UR5

And with the use of the LiDAR sensor to create an external source of odometry for the SLAM process, the measured distance on the 3D model between the two steel strands of the two A and B hollow slabs was 23.48cm and 26.58cm, respectively.

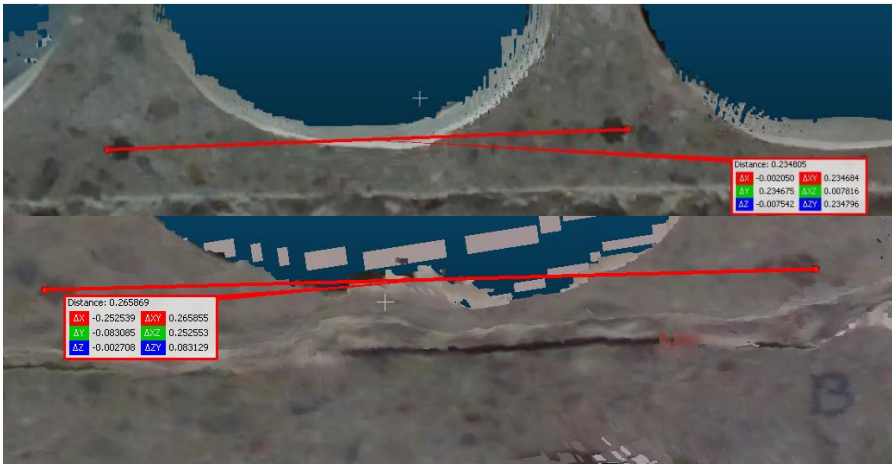


Figure 48. Scanning result with external odometry from 2D LiDAR

Table 1. Summary table of comparisons

	Distance between two steel strands	
	A (cm)	B (cm)
Reference distance	26.5	23.7
Handheld	26.73	23.78
Scanner with UR5 support	26.38	23.51
Scanner with LiDAR odometry	26.58	23.48

6 CONCLUSION AND FUTURE WORK

In this project, a system for performing a reconstruction of an object surface using low-cost devices was developed. The handheld structure with RGB-D and RTABMAP consisted of the basis hardware and software for other expansion design in this project. Extensive solutions based on handheld architecture can be listed such as using pre-processing images data method, storing data from cameras to perform SLAM later, helping the system operate under unstable network conditions and also incorporating 2D LiDAR into the system to provide an external odometry source for RTABMAP can help improve the scan quality when the system was used as a Horizontal planar scanner. In addition, a Cobot was introduced in this project to assist the scanning process using Handheld design where the orbital trajectories of the robotic arm can be planned to increase the automation of the system.

Finally, the entire system is run on a local area network, where all the data flows are sent and communications are made between devices. Therefore, any device or sensor can easily be added to the system to receive or send data that enhances customization and the ability to operate as a system. Generally, the scanning system was built based on the following criteria:

- Quality: The output quality of the scanning system, the reconstructed 3D models of concrete slabs are of high quality with dimensions close to the actual size.
- Mobile: The ability to customize and expand was also one of the top criteria when developing the scanning system. The foundation of the scanning system is the handheld design. However, With minor adjustments, this handheld configuration can be used as a fixed scanning system or a scanning device in combination with Cobot/ Linear sliding support system to increase automation for the system.
- Decentralizing: The system is designed with the general idea that compact devices capable of connecting to the network collect data from sensors and transmit data to the end data processing devices, also receive commands in the opposite direction. So basically, the sensors or Cobot can be operated remotely which is not bound much in the physical structure and connection. And with special system settings, any authorized terminal within the network of the system can receive data from the sensor as well as order the system. Not necessarily a fixed computer at all.
- Automation: The system scans from the start collecting data from the sensor, sending the data to the data processing device and reconstructing 3D model can be done only by the simple start

commands given to the system. In addition, these commands can also be serialized automatically to the system automatically using various feasible methods.

- Price: Regardless of the data processing device, the two devices are the basis for the scanning system are camera and device that collects data stream from camera. The cost of camera used in the system is about 200 euro, and device that capture data from camera with the latest and best version of it priced at about 40 Euro on the market.

3D scanning results in two key things: higher quality products and less costly manufacturing processes. It has been estimated that 3D scanning can reduce manufacturing costs by 75 percent. In checking the final product in the production line, when quality control is achieved, the process must be extremely reliable and as fast as possible. Using a 3D scanner will give any controller the ability to collect complex shape measurements to perform checks by comparing scanned data with a reference 3D model. Based on the measurement result received, user would be able to decide whether parts are accepted or rejected immediately. Thereby, the expensive labor cost for product inspection is minimized with a fully automated scanning system and the modest initial investment. In addition, the system can be used in earlier stages of production in the plant's manufacturing process. In the manufacturing process, finding the root cause as quickly as possible is essential to avoid bottlenecks in the production phase of product lifecycle management. It is necessary to know whether the problem is from the design or the device to get back exactly where it is. The system helps to inspect not merely the products on the production line, but also the tools used in production, for example, frames for precast concrete slabs. It allows getting not only the size easier, but also a better overview of the entire product in one scan. Tool maintenance is critical and mobile 3D scanning is the fastest and most reliable way to test tools at any point in their lifecycle.

In general, with the promising results received from the scanning system from both the technical aspects and practical applicability and the initial investment costs promised for business, the project is now ready to enter further development steps.

Future work

As the automation of the system is always a top priority, the system will be developed towards improving into automation in the future. Automating the process of measuring and comparing the size of a

reconstructed 3D model to the actual desired size to automatically make assessments on each scanned product is the main development direction in the future of the project. This can be done on the basis of Matlab, a programming platform designed specifically for engineers and scientists. Matlab allows establishing links with a web app, which user could use to increase the ability to monitor and manage the systems. Currently, an application designed that runs independently with its own the database performs archival of measurement calculations of each product with each unique ID allowing users to track and retrieve data stored as Excel files.

In the future, one potential development goal is the integration of 3D model presentation tools into this application to increase user interaction. Algorithms for processing automatic measurement of reconstructed models, for instance from Matlab, can also be integrated into this app to make measurements, show them to the operator as well as to automatically store them into the database. On the other hand, an inexpensive automatic sliding system to support the scanning process replacing for Cobot could also be a part of future development plans. Such a support system not only allows the automation of the scanning trajectories of the system, but also helps to increase the motion stability of the scanner, thereby improving the output.

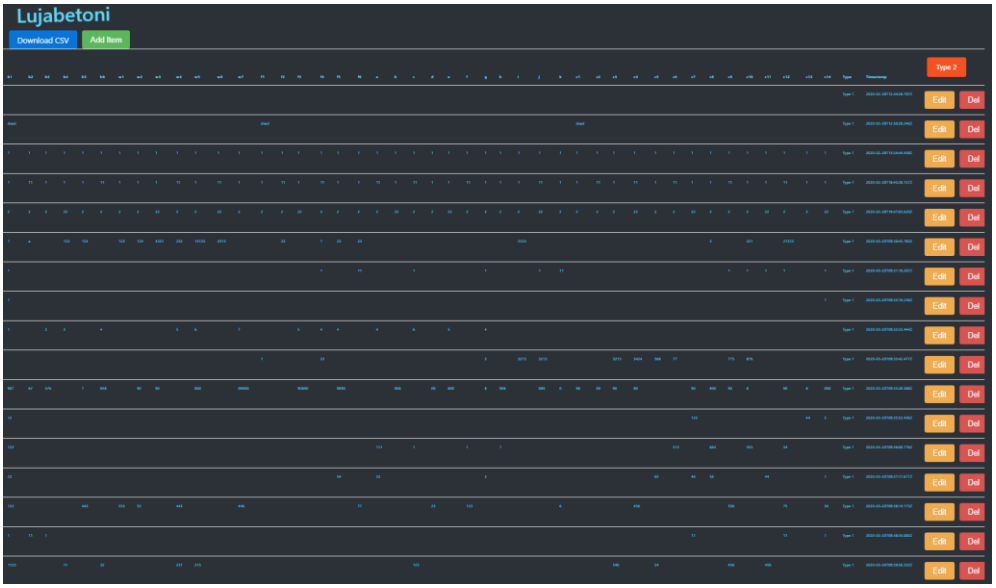


Figure 49. Web app interface

Bibliography

Anil Mahtani, L. S. (n.d.). *Effective Robotics Programming with ROS.*

Bai, X. G. (2012). A GPU accelerated real-time self-contained visual navigation system for UAVs. *Proceedings of the IEEE International Conference on Information and Automation*, (pp. 578–581). Shenyang, China.

Bogue, R. (2018). What are the prospects for robots in the construction industry? *Ind. Robot An Int. J., 45*, 1-6.

Chamberlain, A. (2 - 12, 2012). *Automation and Robotics in Construction XI.* Newnes.

Chen Fu, C. M. (2019). LIDAR and Monocular Camera Fusion: On-road Depth Completion for Autonomous Driving. *2019 IEEE International Conference on Intelligent Transportation Systems* (pp. 273-278). IEEE.

CSM. (n.d.). Retrieved from CSM: https://censi.science/software/csm/

Daniilidis, D. K. (2016). Springer Handbook of Robotics. In D. K. Daniilidis, *Springer Handbook of Robotics* (pp. 811–822). Berlin, Heidelberg.

Drunk, G. (1988). Sensors for Mobile Robots. In *Sensors and Sensory Systems for Advanced Robots.*

Foote, T. (2013). tf: The transform library. *In Proceedings IEEE International Conference on Technologies for Practical Robot Applications, Open-Source Software workshop* (pp. 1–6). IEEE.

Gerald Rauscher, D. D. (2014). A Comparison of 3D Sensors for Wheeled Mobile Robots. In *Intelligent Autonomous Systems 13* (pp. 29– 41).

Hager, H. I. (2016). Springer Handbook of Robotics. In H. I. Hager, *Springer Handbook of Robotics* (pp. 91–112). Heidelberg, Berlin.

I.J. Nagrath, R. M. (2003). *Robotics and Control.* Tata McGraw-Hill Education Pvt. Ltd.

INK, G. T. (n.d.). *toyo-visual*. Retrieved from toyo-visual: https://www.toyo-visual.com/en/products/imgsensor/index.html

International robot standardization within ISO. (n.d.). Retrieved from International robot standardization within ISO: https://ifr.org/standardisation

JFCC. (2013). *Construction Industry Handbook.*

L.H. Forbes, S. A. (Autom. ConStruct., 92 (2018), pp. 297-311). *Modern Construction : Lean Project Delivery and Integrated Practices.* https://doi.org/10.1016/J.AUTCON.2018.04.004.

Labbé, M., & Michaud, F. (Sept 2014). Online global loop closure detection for large-scale multi-session graph-based SLAM. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2661–2666). IEEE.

M. Labbé, a. F. (2018). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*.

Mark W. Spong, S. H. (2005). *Robot Modeling and Control.*

Moqqaddem, S., Ruichek, Y., Touahni, R., & Sbihi, A. (2011). A spectral clustering and kalman filtering based objects detection and tracking using stereo vision with linear cameras. *Intelligent Vehicle, IEEE Symposium* (pp. 902-907). Baden-Baden, Germany: IEEE.

P. Henry, M. K. (2010). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Experimental Robotics.*

Quigley, M. (2009). ROS: an open-source Robot Operating System. *ICRA workshop on open source software*.

Raspberry Pi 4 specs and benchmarks. (n.d.). Retrieved from The MagPi: Raspberry Pi 4 specs and benchmarks

Robert B. Fisher, K. K. (2016). Range Sensors. In *Springer Handbook of Robotics* (pp. 783 – 810). Heidelberg, Berlin: Springer-Verlag.

ROS master. (n.d.). Retrieved from ROS master: http://wiki.ros.org/ROS/Tutorials/MultipleMachines

Ros Wiki. (n.d.). Retrieved from http://wiki.ros.org/imu_filter_madgwick

ROS wiki. (n.d.). Retrieved from http://wiki.ros.org/laser_scan_matcher

Sebastian Thrun, M. M. (May 2006). The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 403-429.

Silfvast, W. T. (2004). *Laser Fundamentals.* Cambridge University Press.

T.D. Oesterreich, F. T. (2016). Understanding the implications of digitisation and automation in the context of Industry 4.0: a triangulation approach and elements of a research agenda for the construction industry. *Computers in Industry (COMPUT IND), 83*, 121-139.

Torreao, J. R. (July 19th 2011). *Advances in Stereo Vision.* IntechOpen.

Vuylsteke, P., & Oosterlinck, A. (1990). Range image acquisition with a single binary-encoded light pattern. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (pp. 148 - 164). IEEE.

X Kang, J. L. (2019). Real-Time RGB-D Simultaneous Localization and Mapping Guided by Terrestrial LiDAR Point Cloud for Indoor 3-D Reconstruction and Camera Pose Estimation.